Advanced Data Structures & Algorithm Analysis

丁尧相
浙江大学

2024.5.20

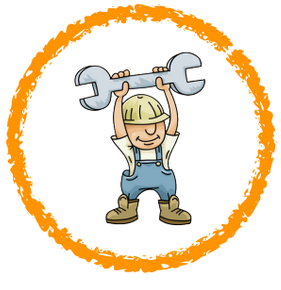
Randomized Algorithms

- Hiring problem: probabilistic analysis
- Randomized hiring algorithm
- Randomized quicksort
- Min-cut and max-cut
- Take-home messages

Randomized Algorithms

- Hiring problem: probabilistic analysis
- Randomized hiring algorithm
- Randomized quicksort
- Min-cut and max-cut
- Take-home messages

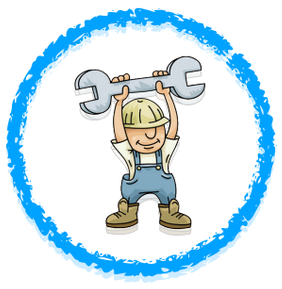
Hiring Problem

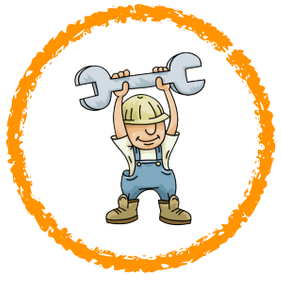


BOSS直聘

Hiring Problem

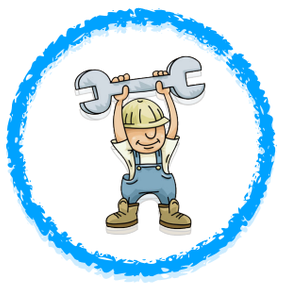
BOSS直聘





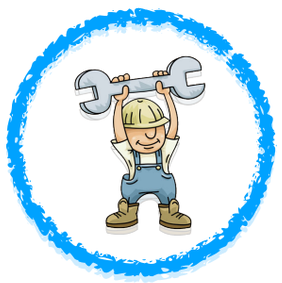
Hiring Problem

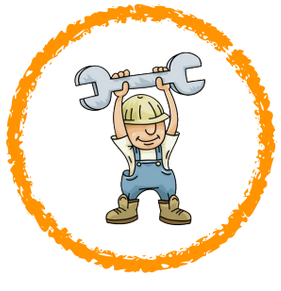
BOSS直聘



Hiring Problem

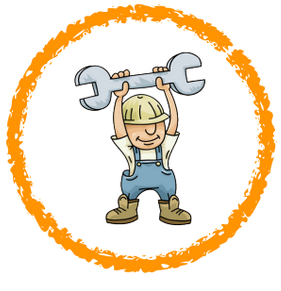
BOSS直聘





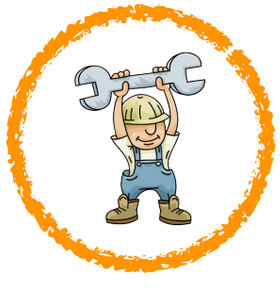
Hiring Problem

BOSS直聘



Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

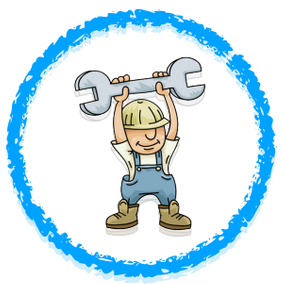
Hiring Problem



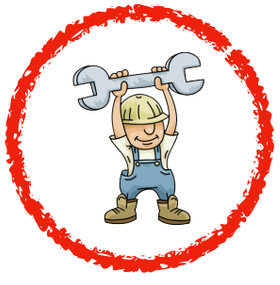
BOSS直聘

Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem



BOSS直聘

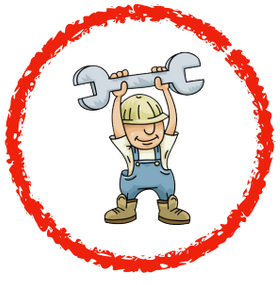


Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem

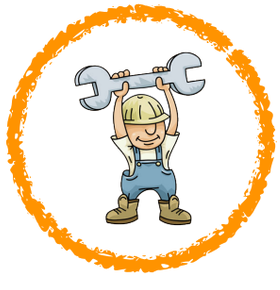
BOSS直聘

Better than current?



Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem



BOSS直聘

Better than current?

Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem

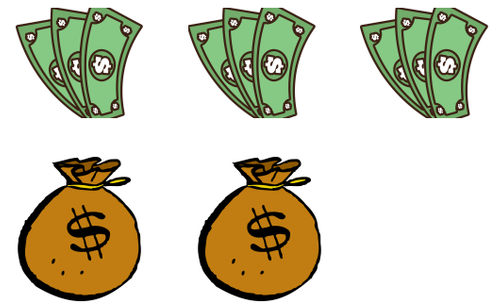
BOSS直聘

Better than current?

Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem

BOSS直聘

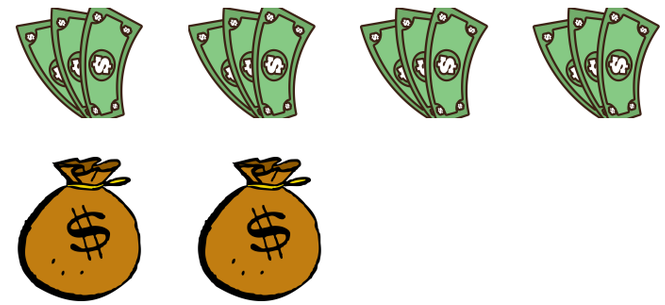


Better than current?

Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem

BOSS直聘

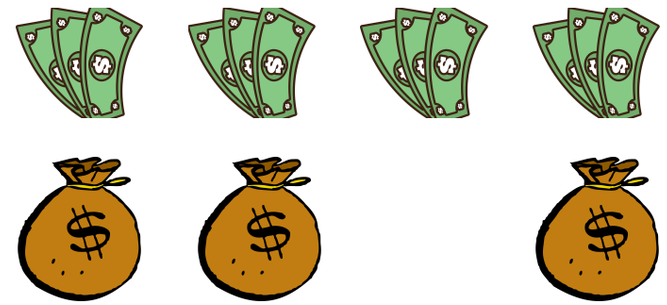


Better than current?

Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem

BOSS直聘

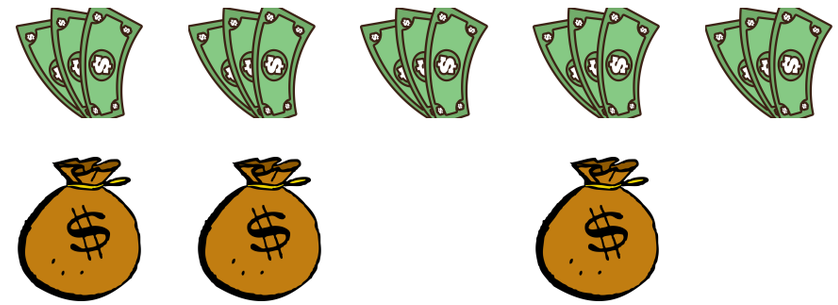


Better than current?

Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem

BOSS直聘

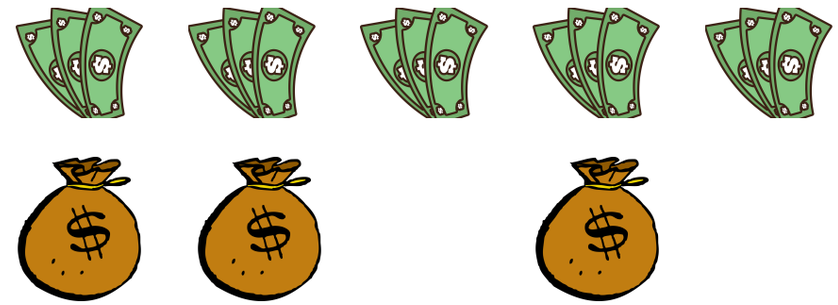


Better than current?

Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem

BOSS直聘



Better than current?

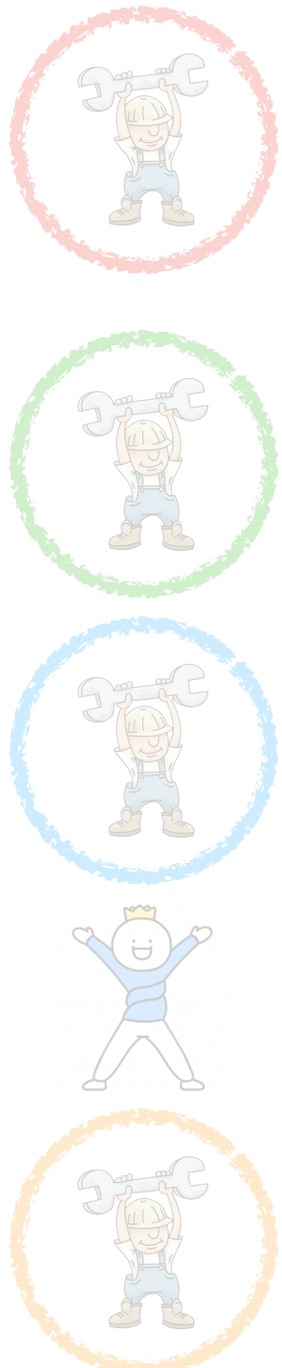
Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

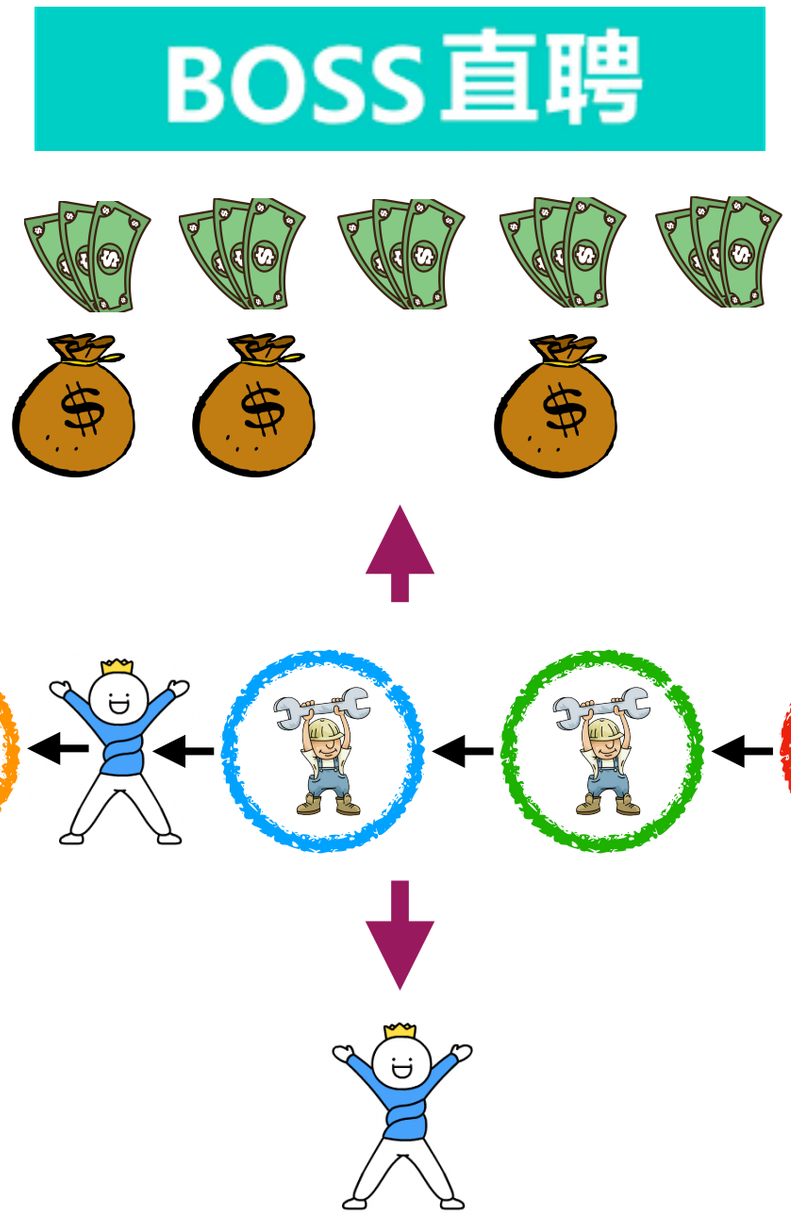
Hiring Problem

Better than current?

Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

Hiring Problem

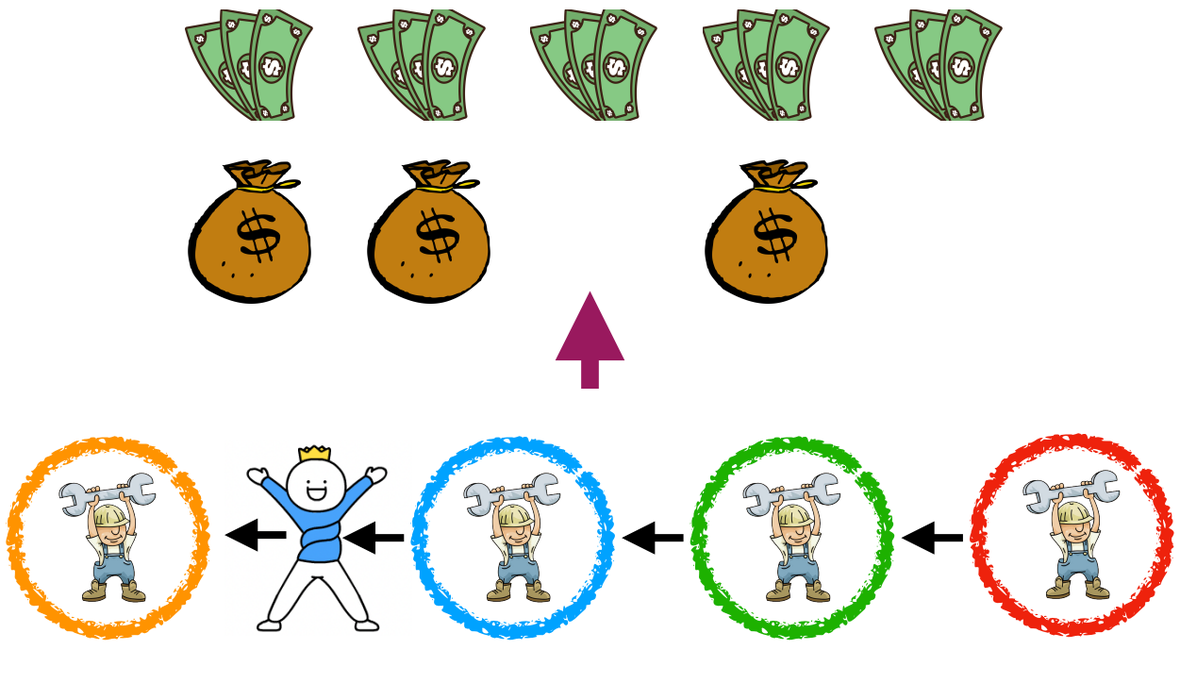




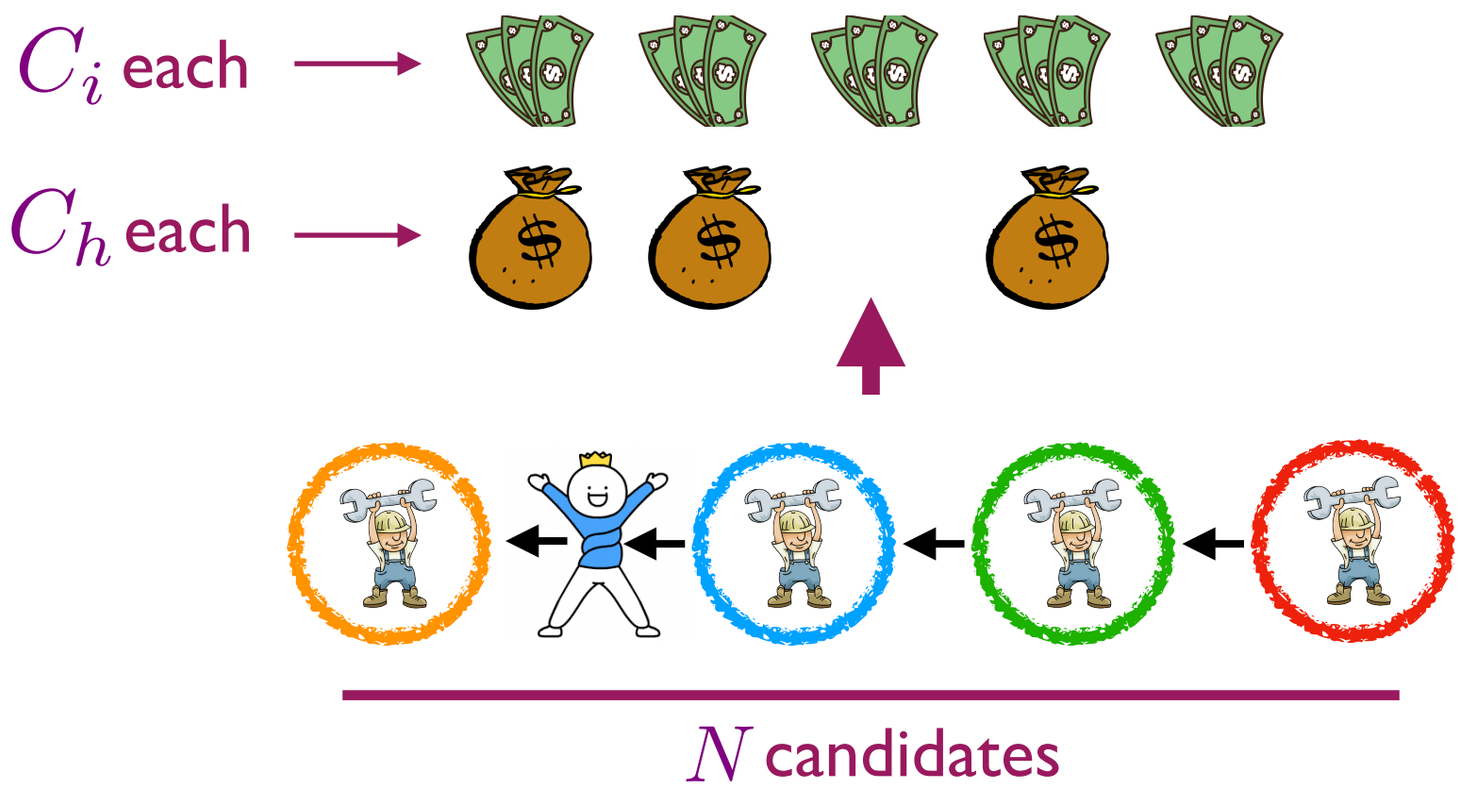
Better than current?

Interview and spend a small cost.
Hire if find better candidate and spend a big cost.

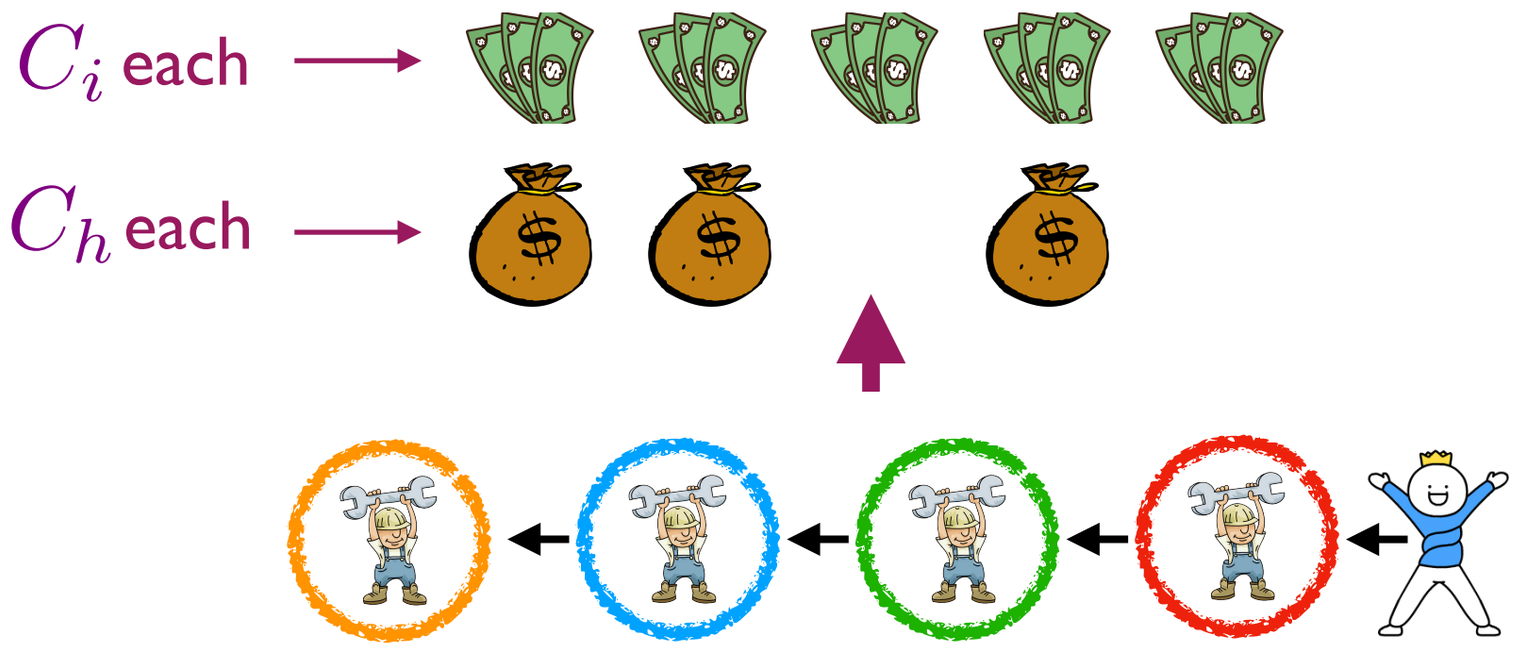
Cost Analysis



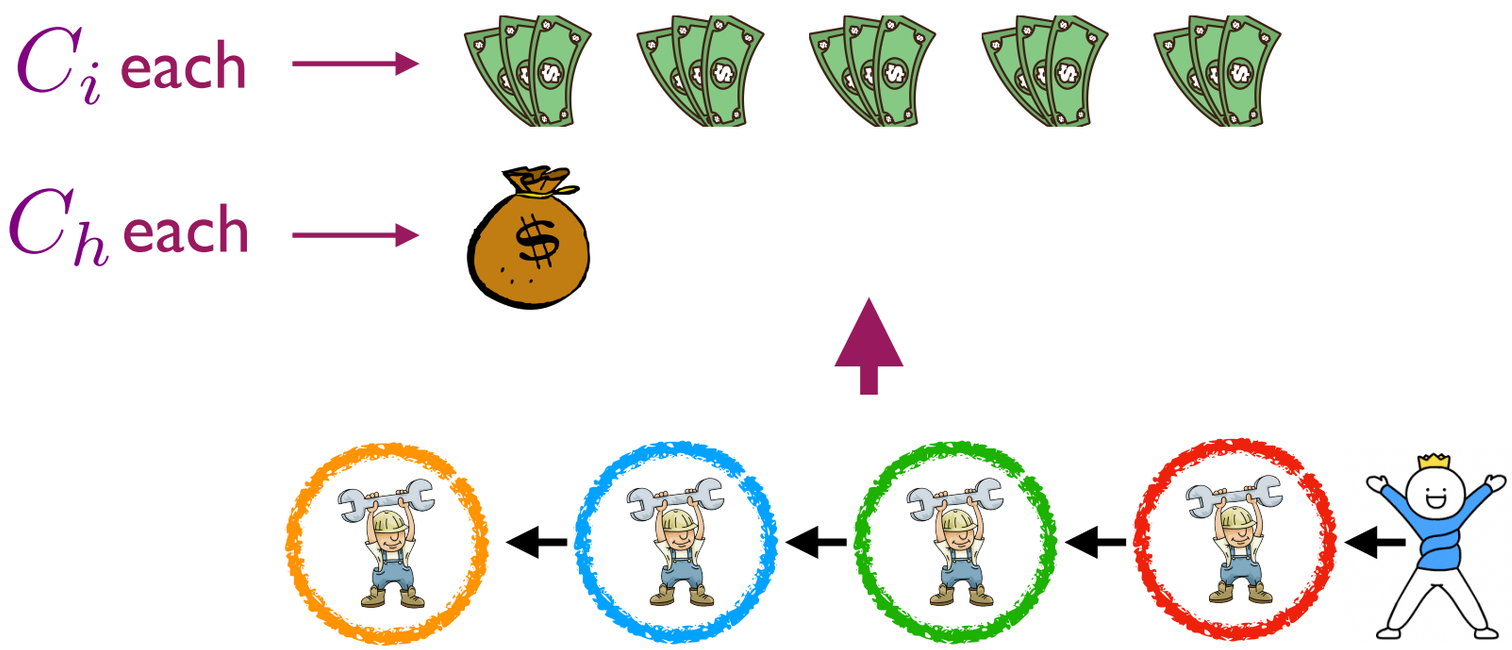
Cost Analysis



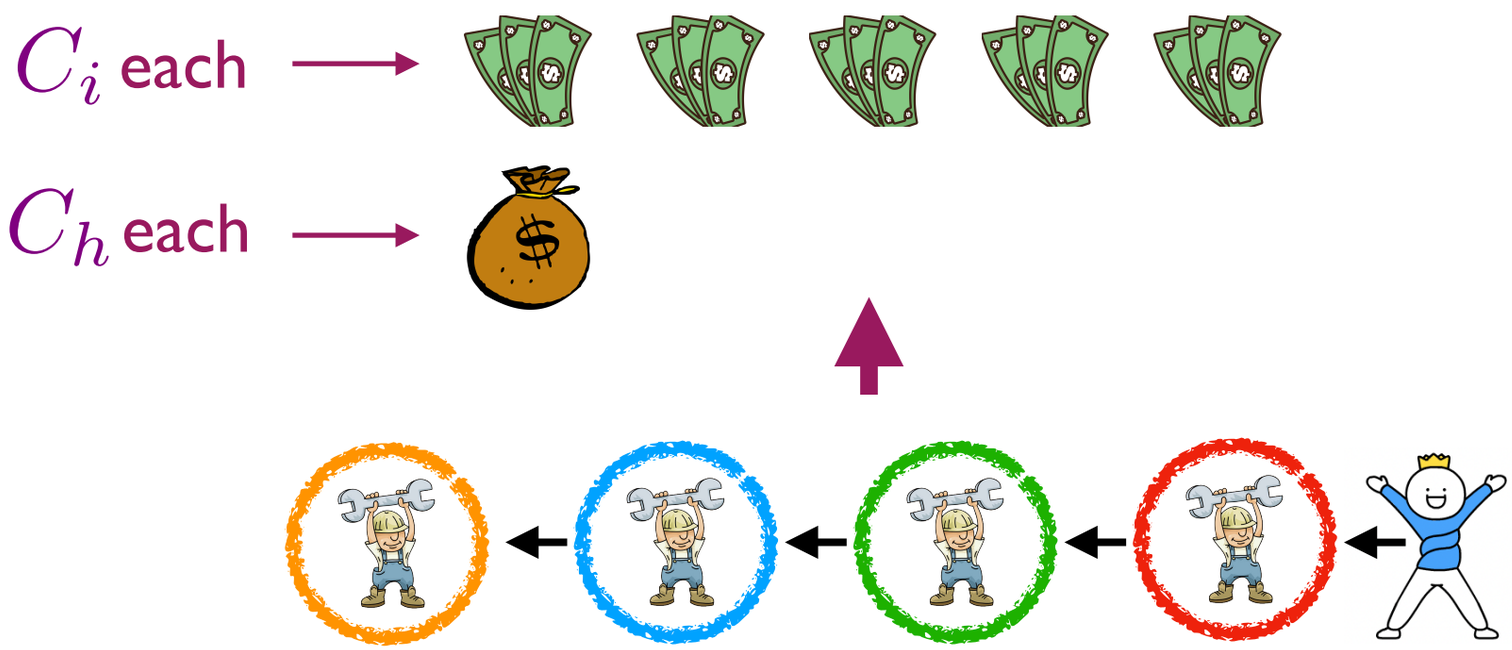
Cost Analysis



Cost Analysis



Cost Analysis



- Time complexity is always $O(N)$.
- More meaningful to consider the total cost on C_h.
- Related to finding the maximum of an array: winner stays.
  But need to minimize the number of winner changes.

Cost Analysis

```
int Hiring ( EventType C[ ], int N )
{   /* candidate 0 is a least-qualified dummy candidate */
    int Best = 0;
    int BestQ = the quality of candidate 0;
    for ( i=1; i<=N; i++ ) {
        Qi = interview( i ); /* Ci */
        if ( Qi > BestQ ) {
            BestQ = Qi;
            Best = i;
            hire( i );  /* Ch */
        }
    }
    return Best;
}
```

Cost Analysis

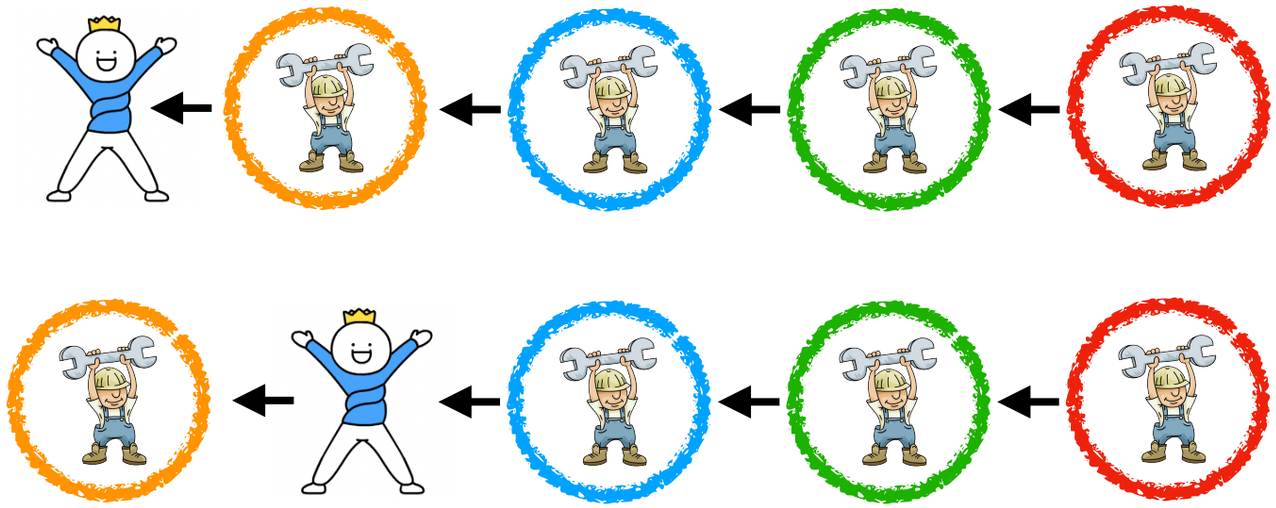
```
int Hiring ( EventType C[ ], int N )
{   /* candidate 0 is a least-qualified dummy candidate */
    int Best = 0;
    int BestQ = the quality of candidate 0;
    for ( i=1; i<=N; i++ ) {
        Qi = interview( i ); /* Ci */
        if ( Qi > BestQ ) {
            BestQ = Qi;
            Best = i;
            hire( i );  /* Ch */
        }
    }
    return Best;
}
```

Cost Analysis

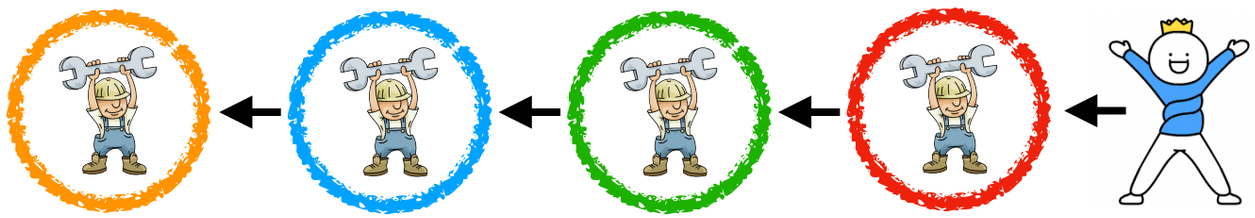
```
int Hiring ( EventType C[ ], int N )
{   /* candidate 0 is a least-qualified dummy candidate */
    int Best = 0;
    int BestQ = the quality of candidate 0;
    for ( i=1; i<=N; i++ ) {
        Qi = interview( i ); /* Ci */
        if ( Qi > BestQ ) {
            BestQ = Qi;
            Best = i;
            hire( i );  /* Ch */
        }
    }
    return Best;
}
```

Unfortunately, the cost is not determined by the algorithm, but the order of input sequence.
Worst case: fully reverse order with NC_h cost.
What is the average cost over all possible orders?

Probabilistic Analysis

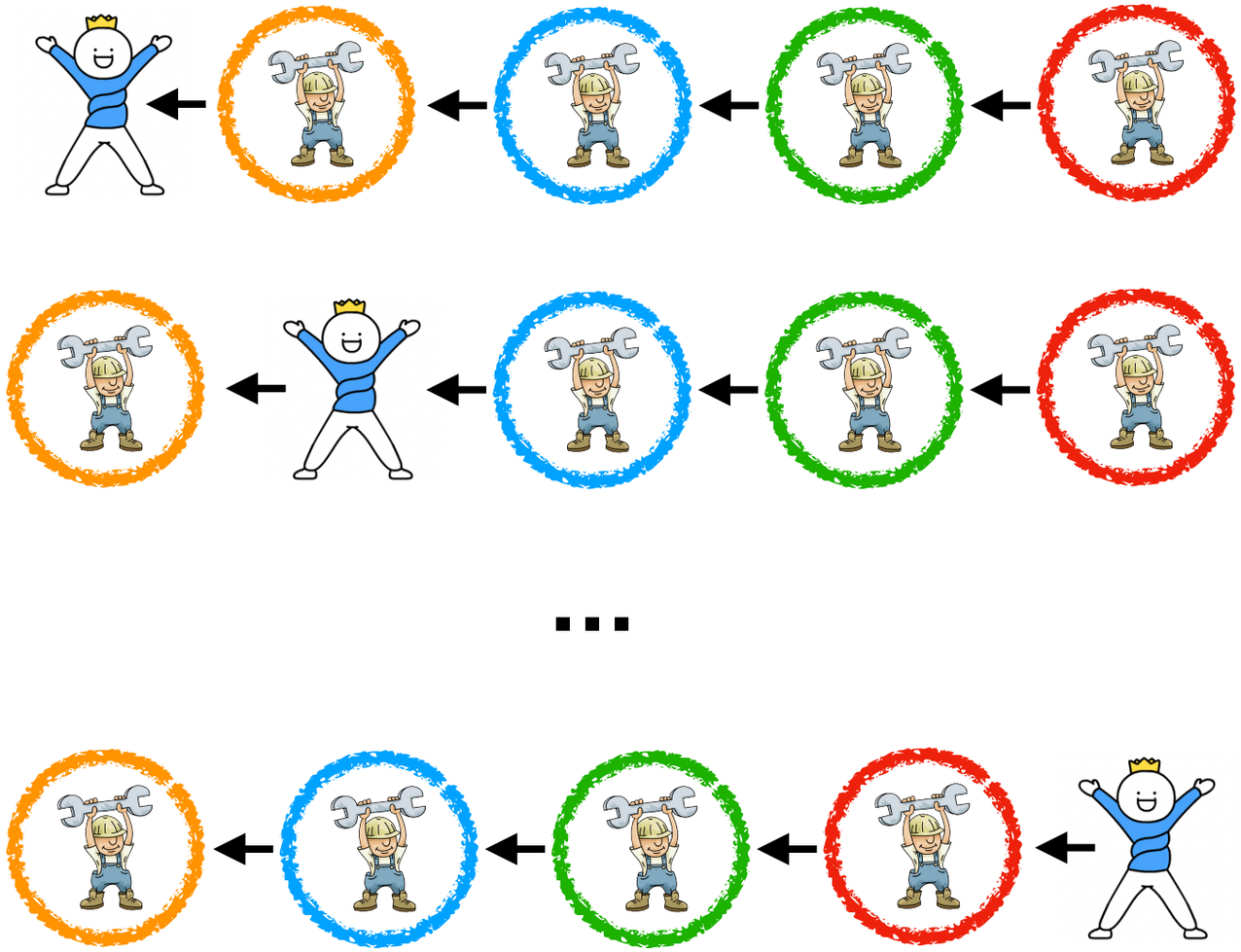


...



$N!$ possible orders

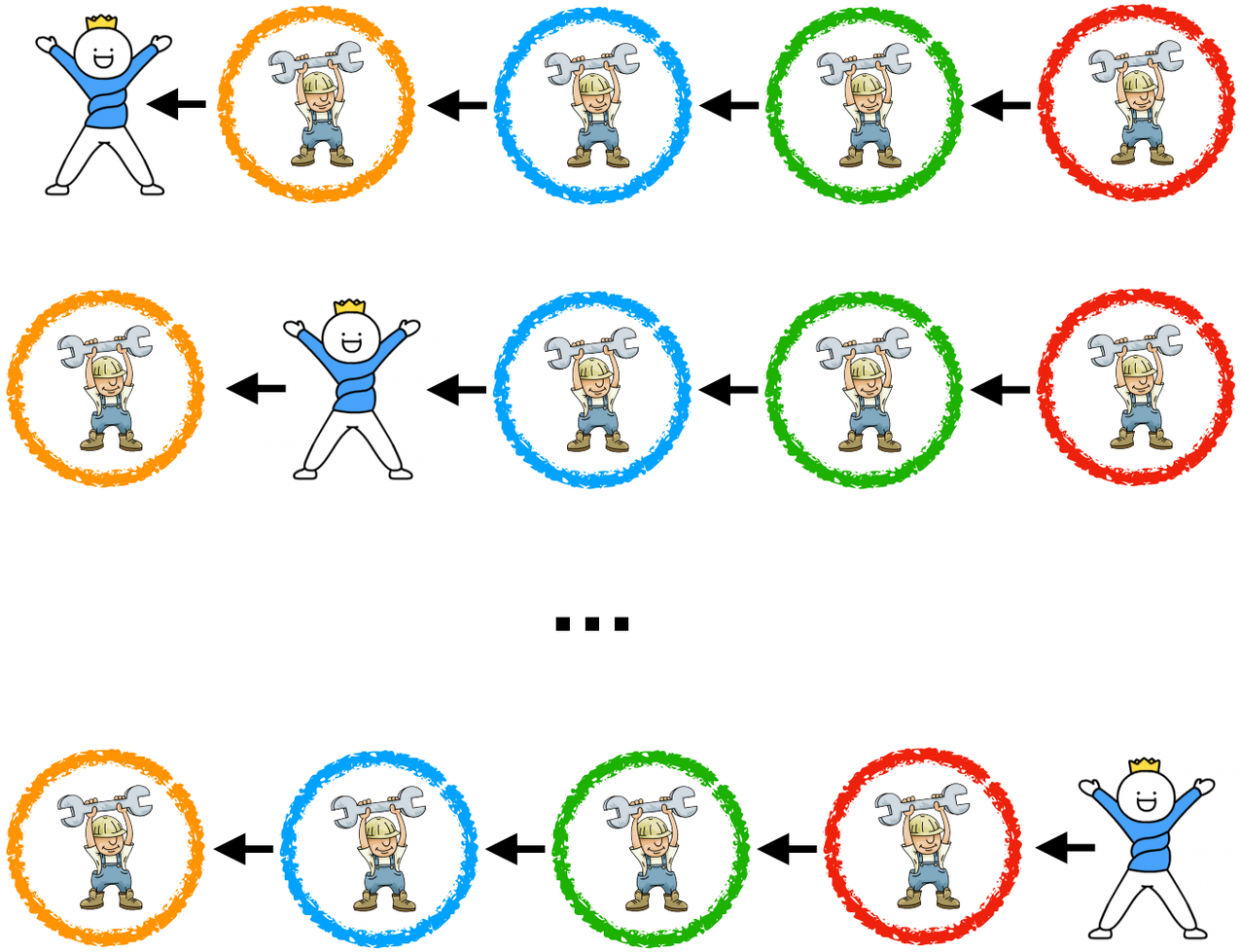
Probabilistic Analysis



...

uniformly draw one from
$N!$ possible orders
randomly as the input.

Probabilistic Analysis



...

uniformly draw one from $N!$ possible orders randomly as the input.

- The "average" cost is the cost in expectation over random sampling.
- Assumption: all inputs can appear with equal chance.

Expectation and Indicator Random Variable

- Expectation of a discrete random variable X : $\mathbb{E}[X] = \sum_{i=1}^{N} x_i \Pr(X = x_i)$

Expectation and Indicator Random Variable

- Expectation of a discrete random variable X : $\mathbb{E}[X] = \sum_{i=1}^{N} x_i \Pr(X = x_i)$

- Linearity of expectation: $\mathbb{E}[\sum_{i=1}^{N} X_i] = \sum_{i=1}^{N} \mathbb{E}[X_i]$

Expectation and Indicator Random Variable

- Expectation of a discrete random variable X : $\mathbb{E}[X] = \sum_{i=1}^{N} x_i \Pr(X = x_i)$

- Linearity of expectation: $\mathbb{E}[\sum_{i=1}^{N} X_i] = \sum_{i=1}^{N} \mathbb{E}[X_i]$

- Indicator random variable of random event A : $\mathbf{I}[A] = \begin{cases} 1, & A \text{ occurs}, \\ 0, & A \text{ doesn't occur}. \end{cases}$

Expectation and Indicator Random Variable

- Expectation of a discrete random variable X : $\mathbb{E}[X] = \sum_{i=1}^{N} x_i \Pr(X = x_i)$

- Linearity of expectation: $\mathbb{E}[\sum_{i=1}^{N} X_i] = \sum_{i=1}^{N} \mathbb{E}[X_i]$

- Indicator random variable of random event A : $\mathbf{I}[A] = \begin{cases} 1, & A \text{ occurs,} \\ 0, & A \text{ doesn't occur.} \end{cases}$

- Expectation of indicator random variable:

$$\mathbb{E}[\mathbf{I}(A)] = 1 \cdot \Pr(A) + 0 \cdot (1 - \Pr(A)) = \Pr(A)$$

Expectation and Indicator Random Variable

- Expectation of a discrete random variable X : $\mathbb{E}[X] = \sum_{i=1}^{N} x_i \Pr(X = x_i)$

- Linearity of expectation: $\mathbb{E}[\sum_{i=1}^{N} X_i] = \sum_{i=1}^{N} \mathbb{E}[X_i]$

- Indicator random variable of random event A : $\mathbf{I}[A] = \begin{cases} 1, & A \text{ occurs,} \\ 0, & A \text{ doesn't occur.} \end{cases}$

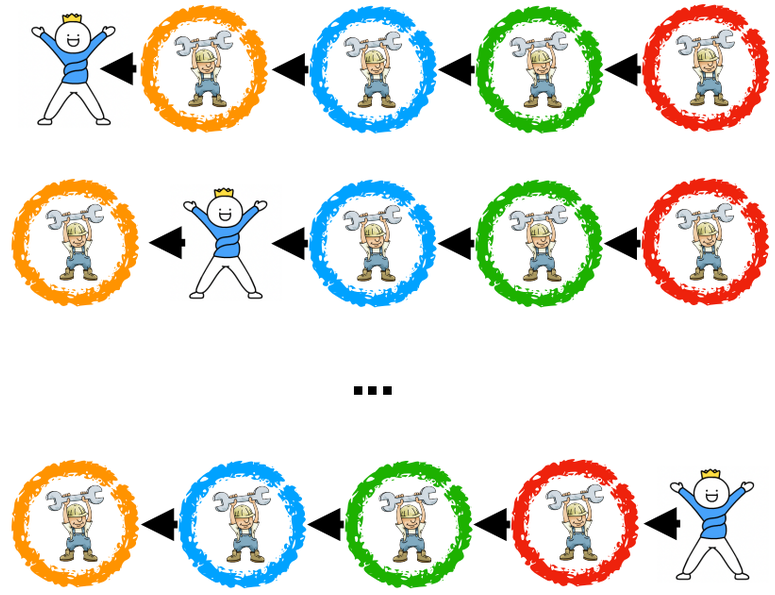
- Expectation of indicator random variable:

$$\mathbb{E}[\mathbf{I}(A)] = 1 \cdot \Pr(A) + 0 \cdot (1 - \Pr(A)) = \Pr(A)$$

Indicator random variable measures whether a specific random event happens.

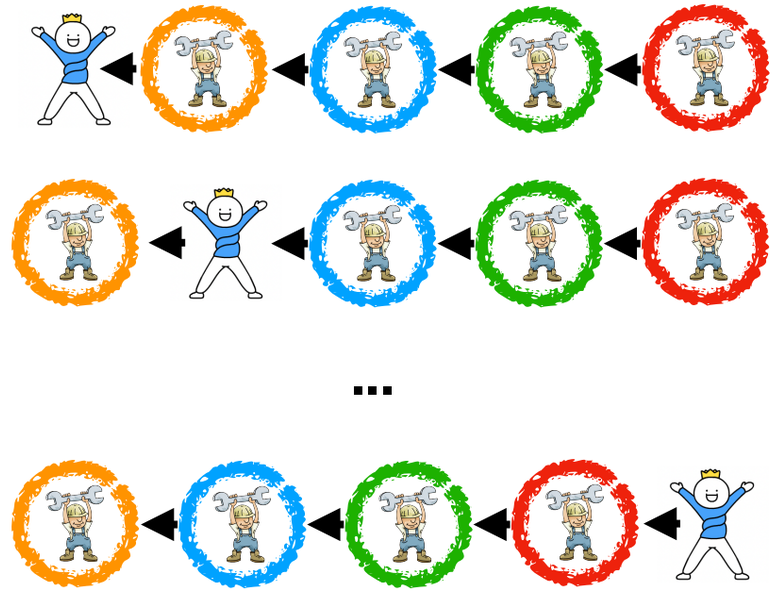
Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.



Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.
- First trial: $\mathbb{E}(C_{total}) = \sum_{i=1}^{N!}(\frac{1}{N!})C_{total,i}$

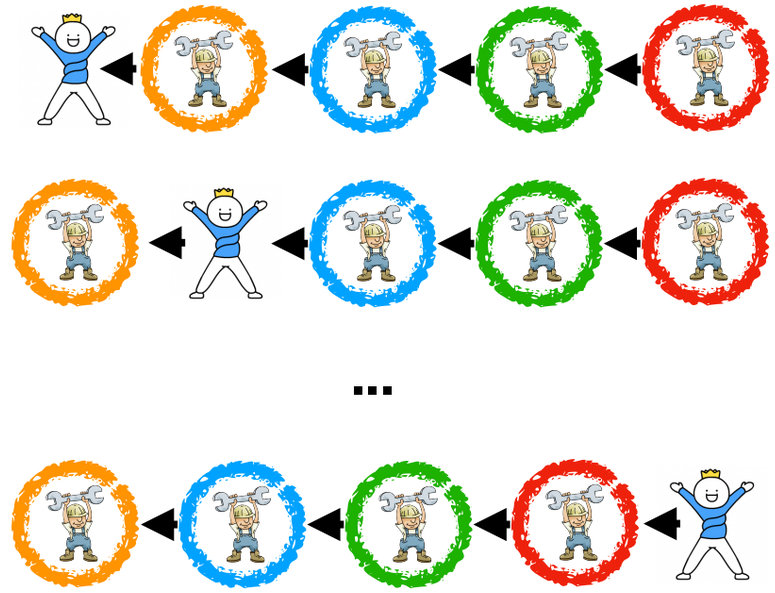


Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.

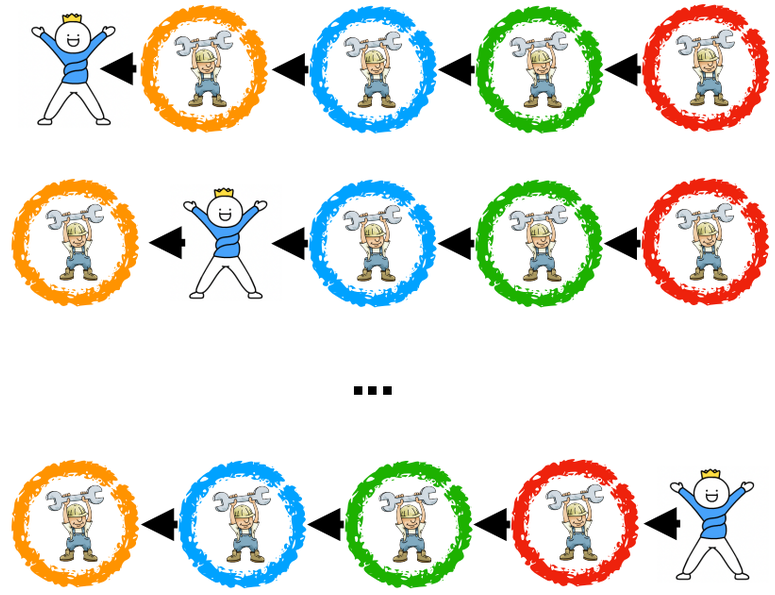
- First trial: $\mathbb{E}(C_{total}) = \sum_{i=1}^{N!}(\frac{1}{N!})\underline{C_{total,i}}$

  The total cost of the i-th order. hard to calculate.



Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.



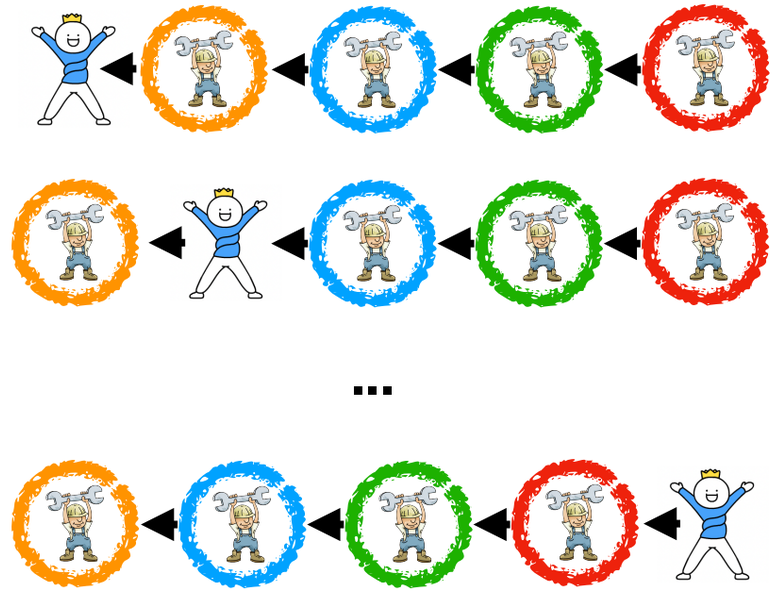
- First trial: $\mathbb{E}(C_{total}) = \sum_{i=1}^{N!}(\frac{1}{N!})\underline{C_{total,i}}$

  The total cost of the i-th order. hard to calculate.

- Second trial: define N random events A_i : whether the i -th candidate is hired.

Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.



- First trial: $\mathbb{E}(C_{total}) = \sum_{i=1}^{N!}(\frac{1}{N!})\underline{C_{total,i}}$

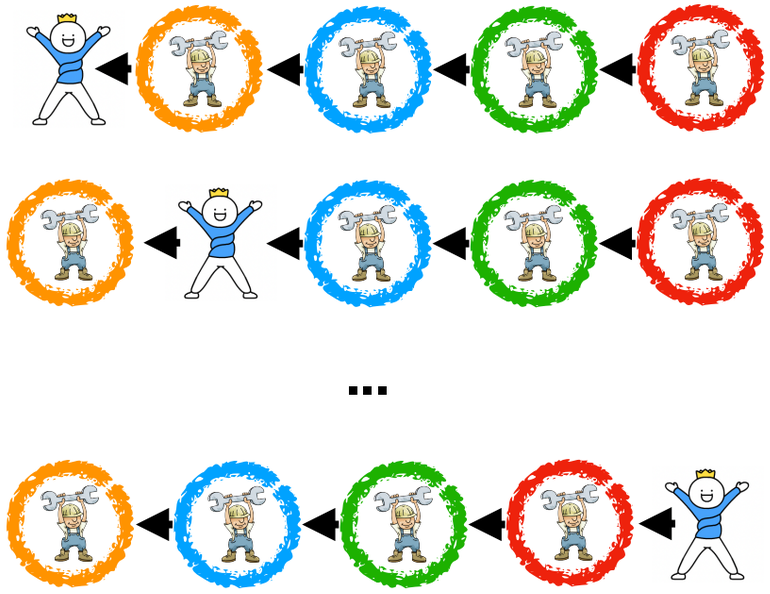
  The total cost of the i-th order. hard to calculate.

- Second trial: define N random events A_i : whether the i -th candidate is hired.

- Define indicator random variable $\mathbf{I}[A_i]$. Then $C_{total} = \sum_{i=1}^{N} \mathbf{I}[A_i]$.

Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.



- First trial: $\mathbb{E}(C_{total}) = \sum_{i=1}^{N!}(\frac{1}{N!})\underline{C_{total,i}}$

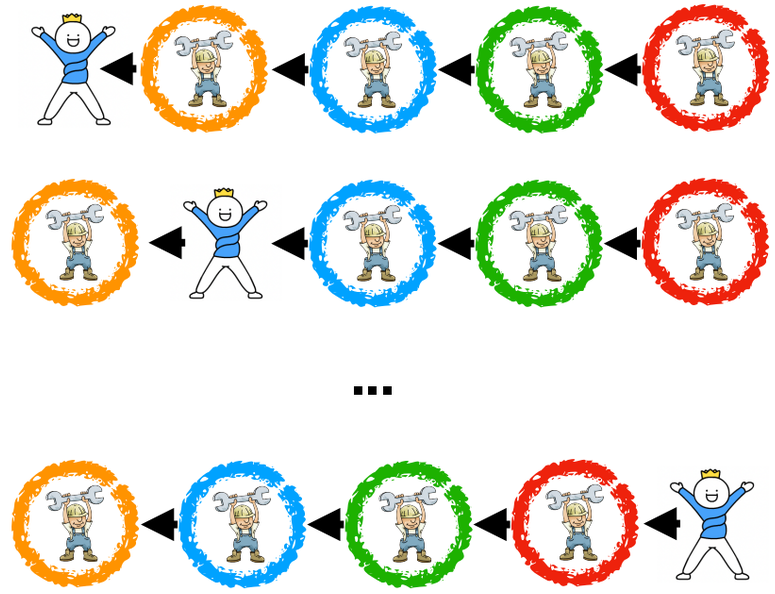
  The total cost of the i-th order. hard to calculate.

- Second trial: define N random events A_i : whether the i -th candidate is hired.

- Define indicator random variable $\mathbf{I}[A_i]$. Then $C_{total} = \sum_{i=1}^{N} \mathbf{I}[A_i]$.
  - $\mathbb{E}[C_{total}] = C_h E[\sum_{i=1}^{N} \mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \mathbb{E}[\mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \Pr(A_i)$

Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.



- First trial: $\mathbb{E}(C_{total}) = \sum_{i=1}^{N!}(\frac{1}{N!})\underline{C_{total,i}}$

  The total cost of the i-th order. hard to calculate.

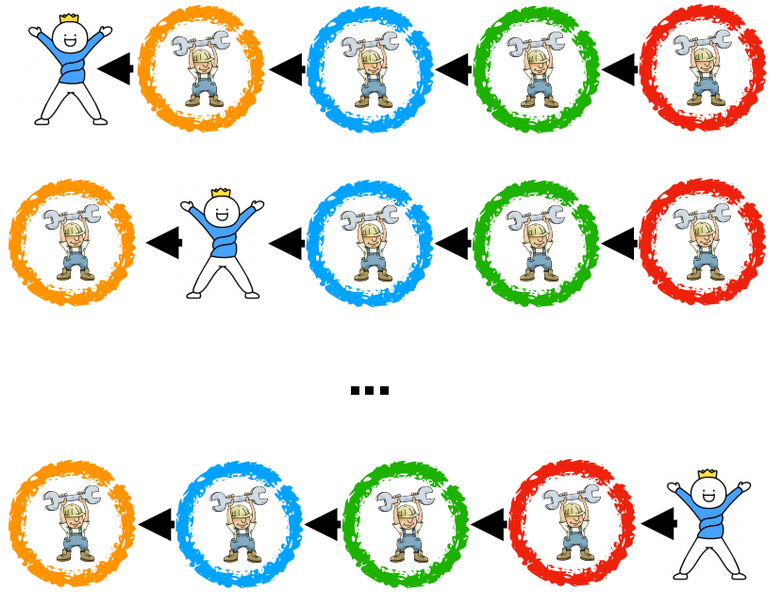
- Second trial: define N random events A_i : whether the i -th candidate is hired.

- Define indicator random variable $\mathbf{I}[A_i]$. Then $C_{total} = \sum_{i=1}^{N} \mathbf{I}[A_i]$.
  - $\mathbb{E}[C_{total}] = C_h E[\sum_{i=1}^{N} \mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \mathbb{E}[\mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \Pr(A_i)$
  - $\Pr(A_i) = 1/i$

The i-the candidate is the best over the first i candidate, while the best has equal chance to appear in any place.

Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.



- First trial: $\mathbb{E}(C_{total}) = \sum_{i=1}^{N!}(\frac{1}{N!})\underline{C_{total,i}}$

  The total cost of the i-th order. hard to calculate.

- Second trial: define N random events A_i : whether the i -th candidate is hired.

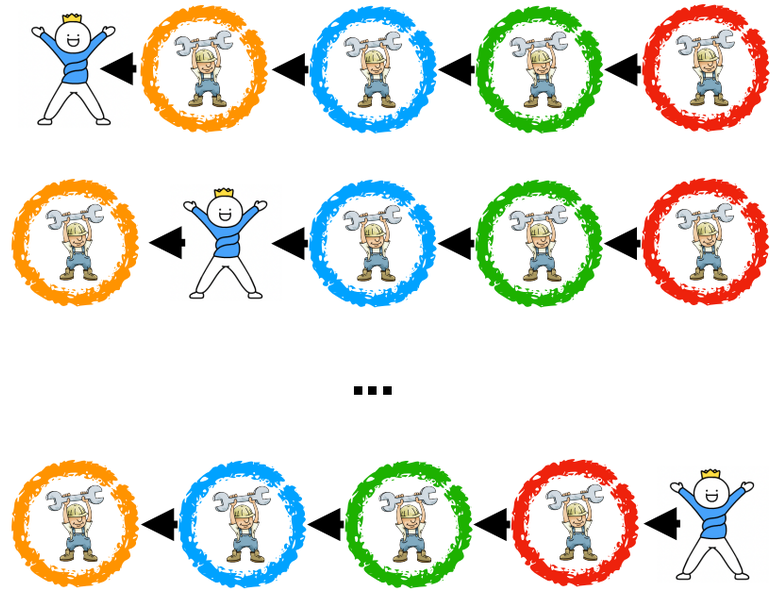
- Define indicator random variable $\mathbf{I}[A_i]$. Then $C_{total} = \sum_{i=1}^{N} \mathbf{I}[A_i]$.
  - $\mathbb{E}[C_{total}] = C_h E[\sum_{i=1}^{N} \mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \mathbb{E}[\mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \Pr(A_i)$
  - $\Pr(A_i) = 1/i$

    The i-the candidate is the best over the first i candidate, while the best has equal chance to appear in any place.

  $\mathbb{E}[C_{total}] = C_h \sum_{i=1}^{N}(1/i) = O(C_h \ln N).$

Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.



- First trial: $\mathbb{E}(C_{total}) = \sum_{i=1}^{N!}(\frac{1}{N!})\underline{C_{total,i}}$

  The total cost of the i-th order. hard to calculate.

- Second trial: define N random events A_i : whether the i -th candidate is hired.

- Define indicator random variable $\mathbf{I}[A_i]$. Then $C_{total} = \sum_{i=1}^{N} \mathbf{I}[A_i]$.
  - $\mathbb{E}[C_{total}] = C_h E[\sum_{i=1}^{N} \mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \mathbb{E}[\mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \Pr(A_i)$
  - $\Pr(A_i) = 1/i$

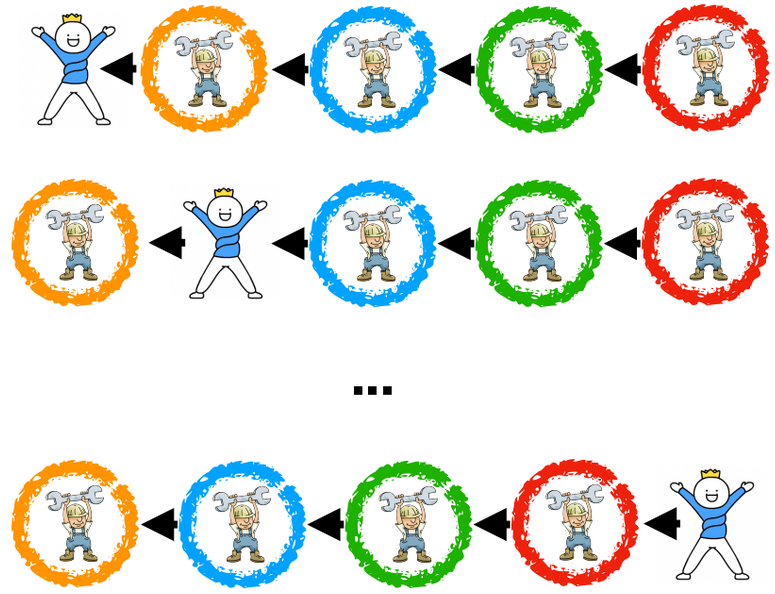
    The i-the candidate is the best over the first i candidate, while the best has equal chance to appear in any place.

    $\mathbb{E}[C_{total}] = C_h \sum_{i=1}^{N}(1/i) = O(C_h \ln N).$

    The power of choosing the right random event.
    Great importance in randomized algorithm design and analysis.

Prob. Analysis of Hiring Problem

- Assume uniform distribution over all $N!$ orders. Calculate cost in expectation.



- First trial: $\mathbb{E}(C_{total}) = \sum_{i=1}^{N!}(\frac{1}{N!})\underline{C_{total,i}}$

  The total cost of the i-th order. hard to calculate.

- Second trial: define N random events A_i : whether the i -th candidate is hired.

- Define indicator random variable $\mathbf{I}[A_i]$. Then $C_{total} = \sum_{i=1}^{N} \mathbf{I}[A_i]$.
  - $\mathbb{E}[C_{total}] = C_h E[\sum_{i=1}^{N} \mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \mathbb{E}[\mathbf{I}[A_i]] = C_h \sum_{i=1}^{N} \Pr(A_i)$
  - $\Pr(A_i) = 1/i$

    The i-the candidate is the best over the first i candidate, while the best has equal chance to appear in any place.

    $\mathbb{E}[C_{total}] = C_h \sum_{i=1}^{N}(1/i) = O(C_h \ln N).$

    The power of choosing the right random event.
    Great importance in randomized algorithm design and analysis.

Can we utilize randomness in algorithm design?

Randomized Algorithms

- Hiring problem: probabilistic analysis
- Randomized hiring algorithm
- Randomized quicksort
- Min-cut and max-cut
- Take-home messages

Randomized Hiring Algorithm

```
int RandomizedHiring ( EventType C[ ], int N )
{   /* candidate 0 is a least-qualified dummy candidate */
    int Best = 0;
    int BestQ = the quality of candidate 0;

    randomly permute the list of candidates;

    for ( i=1; i<=N; i++ ) {
        Qi = interview( i ); /* Ci */
        if ( Qi > BestQ ) {
            BestQ = Qi;
            Best = i;
            hire( i );  /* Ch */
        }
    }
```

Randomized Hiring Algorithm

```
int RandomizedHiring ( EventType C[ ], int N )
{   /* candidate 0 is a least-qualified dummy candidate */
    int Best = 0;
    int BestQ = the quality of candidate 0;

    randomly permute the list of candidates;

    for ( i=1; i<=N; i++ ) {
        Qi = interview( i ); /* Ci */
        if ( Qi > BestQ ) {
            BestQ = Qi;
            Best = i;
            hire( i );  /* Ch */
        }
    }
```

the only change in the algorithm

Randomized Hiring Algorithm

```
int RandomizedHiring ( EventType C[ ], int N )
{   /* candidate 0 is a least-qualified dummy candidate */
    int Best = 0;
    int BestQ = the quality of candidate 0;

    randomly permute the list of candidates;

    for ( i=1; i<=N; i++ ) {
        Qi = interview( i ); /* Ci */
        if ( Qi > BestQ ) {
            BestQ = Qi;
            Best = i;
            hire( i );  /* Ch */
        }
    }
```

the only change in the algorithm

Obviously, the average cost remains the same to the original algorithm.
While even for the worst input case, we can expect the average cost!
In many problems, randomization is a strong technique to reduce the worst-case complexity.

Radomized Permutation Algorithm

Radomized Permutation Algorithm

Target : Permute array A[ ]

Radomized Permutation Algorithm

Target : Permute array A[ ]

Assign each element A[i] a *random priority* P[i],

and sort

Radomized Permutation Algorithm

Target : Permute array A[ ]

Assign each element A[i] a *random priority* P[i],

and sort

```
void PermuteBySorting ( ElemType A[ ], int N )
{
   for ( i=1; i<=N; i++ )
      A[i].P = 1 + rand()%(N^3);
      /* makes it more likely that all priorities are unique */
   Sort A, using P as the sort keys;
}
```

Radomized Permutation Algorithm

Target : Permute array A[ ]

Assign each element A[i] a *random priority* P[i],

and sort

```
void PermuteBySorting ( ElemType A[ ], int N )
{
   for ( i=1; i<=N; i++ )
      A[i].P = 1 + rand()%(N^3);
      /* makes it more likely that all priorities are unique */
   Sort A, using P as the sort keys;
}
```

Claim: **PermuteBySorting produces a *uniform random permutation* of the input, assuming all priorities are distinct.**

Online Hiring Algorithm – hire only once

Online Hiring Algorithm – hire only once

```
int OnlineHiring ( EventType C[ ], int N, int k )
{
    int Best = N;
    int BestQ = - ∞ ;
    for ( i=1; i<=k; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ )   BestQ = Qi;
    }
    for ( i=k+1; i<=N; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ ) {
            Best = i;
            break;
        }
    }
    return Best;
}
```

Online Hiring Algorithm – hire only once

```
int OnlineHiring ( EventType C[ ], int N, int k )
{
    int Best = N;
    int BestQ = - ∞ ;
    for ( i=1; i<=k; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ )   BestQ = Qi;
    }
    for ( i=k+1; i<=N; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ ) {
            Best = i;
            break;
        }
    }
    return Best;
}
```

Online Hiring Algorithm – hire only once

```
int OnlineHiring ( EventType C[ ], int N, int k )
{
    int Best = N;
    int BestQ = - ∞ ;
    for ( i=1; i<=k; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ )   BestQ = Qi;
    }
    for ( i=k+1; i<=N; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ ) {
            Best = i;
            break;
        }
    }
    return Best;
}
```

☞ **What is the probability**

we hire the best qualified candidate for a given *k*?

Online Hiring Algorithm – hire only once

```
int OnlineHiring ( EventType C[ ], int N, int k )
{
    int Best = N;
    int BestQ = - ∞ ;
    for ( i=1; i<=k; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ )   BestQ = Qi;
    }
    for ( i=k+1; i<=N; i++ ) {
        Qi = interview( i );
        if ( Qi > BestQ ) {
            Best = i;
            break;
        }
    }
    return Best;
}
```

☞ What is the **probability** we hire the best qualified candidate for a given k?

☞ What is the **best value of** k to maximize above probability?

Online Hiring Algorithm

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?

{ A:= the best one is at position i }

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?

$$\{ \text{A:= the best one is at position } i \}$$
$$\cap \{ \text{B:= no one at positions } k+1 \sim i-1 \text{ are hired} \}$$

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?

independent

{ A:= the best one is at position i }
$\cap$ { B:= no one at positions k+1 ~ i–1 are hired }

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?

independent

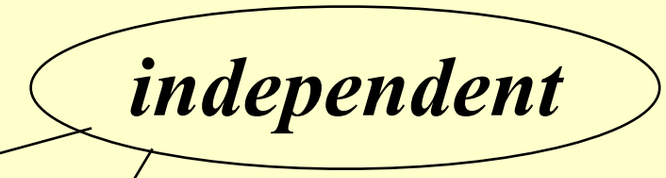
{ A:= the best one is at position i }
$\cap$ { B:= no one at positions k+1 ~ i–1 are hired }

$$\Pr[S_i] = \Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$$

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?



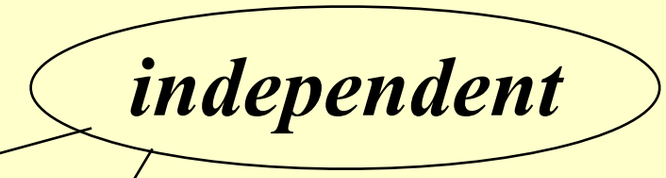
{ A:= the best one is at position i }
$\cap$ { B:= no one at positions k+1 ~ i–1 are hired }

$$\Pr[S_i] = \Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$$
$$= 1/N$$

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?



{ A:= the best one is at position i }
$\cap$ { B:= no one at positions k+1 ~ i–1 are hired }

$$\Pr[S_i] = \Pr[A \cap B] = \Pr[A] \cdot \Pr[B]$$
$$= 1/N \quad = k/(i-1)$$

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?

independent

{ A:= the best one is at position i }
$\cap$ { B:= no one at positions k+1 ~ i–1 are hired }

$$\Pr[S_i] = \Pr[A \cap B] = \Pr[A] \cdot \Pr[B] = \frac{k}{N(i-1)}$$
$$= 1/N \quad = k/(i-1)$$

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?

independent

{ A:= the best one is at position i }
$\cap$ { B:= no one at positions k+1 ~ i–1 are hired }

$$\Pr[S_i] = \Pr[A \cap B] = \Pr[A] \cdot \Pr[B] = 1/N = k/(i-1) = \frac{k}{N(i-1)}$$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?

independent

{ A:= the best one is at position i }
$\cap$ { B:= no one at positions k+1 ~ i–1 are hired }

$$\Pr[S_i] = \Pr[A \cap B] = \Pr[A] \cdot \Pr[B] = 1/N = k/(i-1) = \frac{k}{N(i-1)}$$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

Online Hiring Algorithm

S_i := the ith applicant is the best and is hired

What needs to happen for S_i to be TRUE?

independent

{ A:= the best one is at position i }
$\cap$ { B:= no one at positions k+1 ~ i–1 are hired }

$$\Pr[S_i] = \Pr[A \cap B] = \Pr[A] \cdot \Pr[B] = 1/N = k/(i-1) = \frac{k}{N(i-1)}$$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

Discussion 18: Prove $\int_k^N \frac{1}{x} dx \le \sum_{i=k}^{N-1} \frac{1}{i} \le \int_{k-1}^{N-1} \frac{1}{x} dx$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

☞ **What is the probability we hire the best qualified candidate for a given k?**

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

☞ **What is the probability we hire the best qualified candidate for a given k?**

$$\frac{k}{N} \ln\left(\frac{N}{k}\right) \le \Pr[S] \le \frac{k}{N} \ln\left(\frac{N-1}{k-1}\right)$$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

☞ **What is the probability we hire the best qualified candidate for a given k?**

$$\frac{k}{N} \ln\left(\frac{N}{k}\right) \le \Pr[S] \le \frac{k}{N} \ln\left(\frac{N-1}{k-1}\right)$$

☞ **What is the best value of k to maximize the above probability?**

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

☞ **What is the probability we hire the best qualified candidate for a given k?**

$$\frac{k}{N} \ln\left(\frac{N}{k}\right) \le \Pr[S] \le \frac{k}{N} \ln\left(\frac{N-1}{k-1}\right)$$

☞ **What is the best value of k to maximize the above probability?**

Discussion 19: What is the maximum value of $f(k) = \frac{k}{N} \ln\left(\frac{N}{k}\right)$? And what is the best k?

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

☞ **What is the probability we hire the best qualified candidate for a given k?**

$$\frac{k}{N} \ln\left(\frac{N}{k}\right) \le \Pr[S] \le \frac{k}{N} \ln\left(\frac{N-1}{k-1}\right)$$

☞ **What is the best value of k to maximize the above probability?**

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

☞ **What is the probability we hire the best qualified candidate for a given k?**

$$\frac{k}{N} \ln\left(\frac{N}{k}\right) \le \Pr[S] \le \frac{k}{N} \ln\left(\frac{N-1}{k-1}\right)$$

☞ **What is the best value of k to maximize the above probability?**

$$\frac{d}{dk}\left[\frac{k}{N} \ln\left(\frac{N}{k}\right)\right] = \frac{1}{N}(\ln N - \ln k - 1) = 0$$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

☞ **What is the probability we hire the best qualified candidate for a given k?**

$$\frac{k}{N} \ln\left(\frac{N}{k}\right) \le \Pr[S] \le \frac{k}{N} \ln\left(\frac{N-1}{k-1}\right)$$

☞ **What is the best value of k to maximize the above probability?**

$$\frac{d}{dk}\left[\frac{k}{N} \ln\left(\frac{N}{k}\right)\right] = \frac{1}{N}(\ln N - \ln k - 1) = 0 \quad \Rightarrow \ln k = \ln N - 1 \quad \Rightarrow k = \frac{N}{e}$$

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

☞ **What is the probability we hire the best qualified candidate for a given k?**

$$\frac{k}{N} \ln\left(\frac{N}{k}\right) \le \Pr[S] \le \frac{k}{N} \ln\left(\frac{N-1}{k-1}\right)$$

☞ **What is the best value of k to maximize the above probability?**

$$\frac{d}{dk}\left[\frac{k}{N} \ln\left(\frac{N}{k}\right)\right] = \frac{1}{N}(\ln N - \ln k - 1) = 0 \quad \Rightarrow \ln k = \ln N - 1 \quad \Rightarrow k = \frac{N}{e}$$

☞ **Succeed in hiring the best-qualified applicant with probability at least**

$$\Pr[S] = \sum_{i=k+1}^{N} \Pr[S_i] = \sum_{i=k+1}^{N} \frac{k}{N(i-1)} = \frac{k}{N} \sum_{i=k}^{N-1} \frac{1}{i}$$

☞ **What is the probability we hire the best qualified candidate for a given k?**

$$\frac{k}{N} \ln\left(\frac{N}{k}\right) \le \Pr[S] \le \frac{k}{N} \ln\left(\frac{N-1}{k-1}\right)$$

☞ **What is the best value of k to maximize the above probability?**

$$\frac{d}{dk}\left[\frac{k}{N} \ln\left(\frac{N}{k}\right)\right] = \frac{1}{N}(\ln N - \ln k - 1) = 0 \quad \Rightarrow \ln k = \ln N - 1 \quad \Rightarrow k = \frac{N}{e}$$

☞ **Succeed in hiring the best-qualified applicant with probability at least $1/e$**

Randomization in Algorithm Design

Randomized algorithms has steps where randomness are injected, resulted in two types of algorithms:

- Monte-Carlo algorithm:
  - Deterministic running time
  - Return correct result with probability

- Las Vegas algorithm:
  - Randomized running time
  - Always return the correct result

Relaxation is necessary for dealing with hard computation problems.

Randomized Algorithms

- Hiring problem: probabilistic analysis
- Randomized hiring algorithm
- Randomized quicksort
- Min-cut and max-cut
- Take-home messages

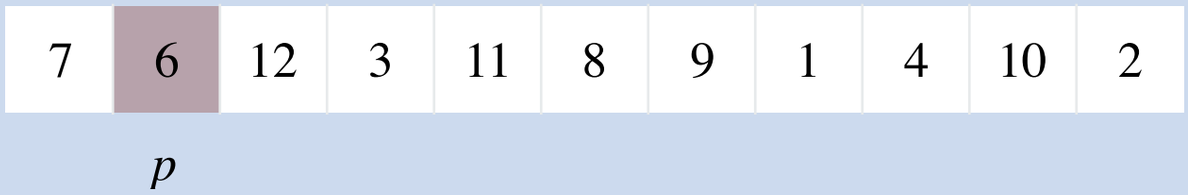
3-WAY PARTITIONING

Goal. Given an array A and pivot element p, partition array so that:

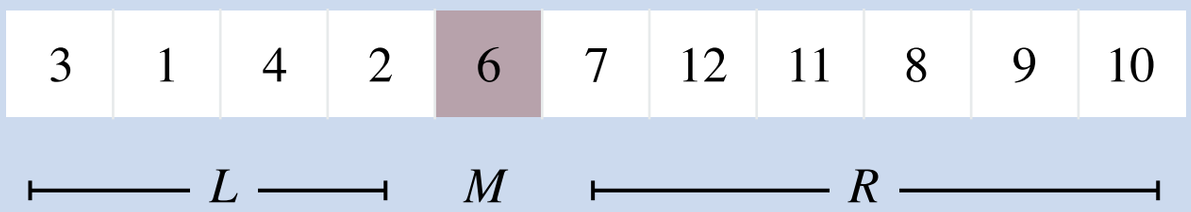
- Smaller elements in left subarray L.
- Equal elements in middle subarray M.
- Larger elements in right subarray R.

Challenge. $O(n)$ time and $O(1)$ space.

the array A



the partitioned array A



Randomized quicksort

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L, M, and R.
- Recursively sort both L and R.

the array A	7	6	12	3	11	8	9	1	4	10	2
		p									

partition A	3	1	4	2	6	7	12	11	8	9	10
sort L	1	2	3	4	6	7	12	11	8	9	10
sort R	1	2	3	4	6	7	8	9	10	11	12
the sorted array A	1	2	3	4	6	7	8	9	10	11	12

Randomized quicksort

- Pick a random pivot element $p \in A$.
- 3-way partition the array into L, M, and R.
- Recursively sort both L and R.

RANDOMIZED-QUICKSORT(A)

IF (array A has zero or one element)

  RETURN.

Pick pivot $p \in A$ uniformly at random.

$(L, M, R) \leftarrow$ PARTITION-3-WAY(A, p). ← $\Theta(n)$

RANDOMIZED-QUICKSORT(L). ← $T(i)$

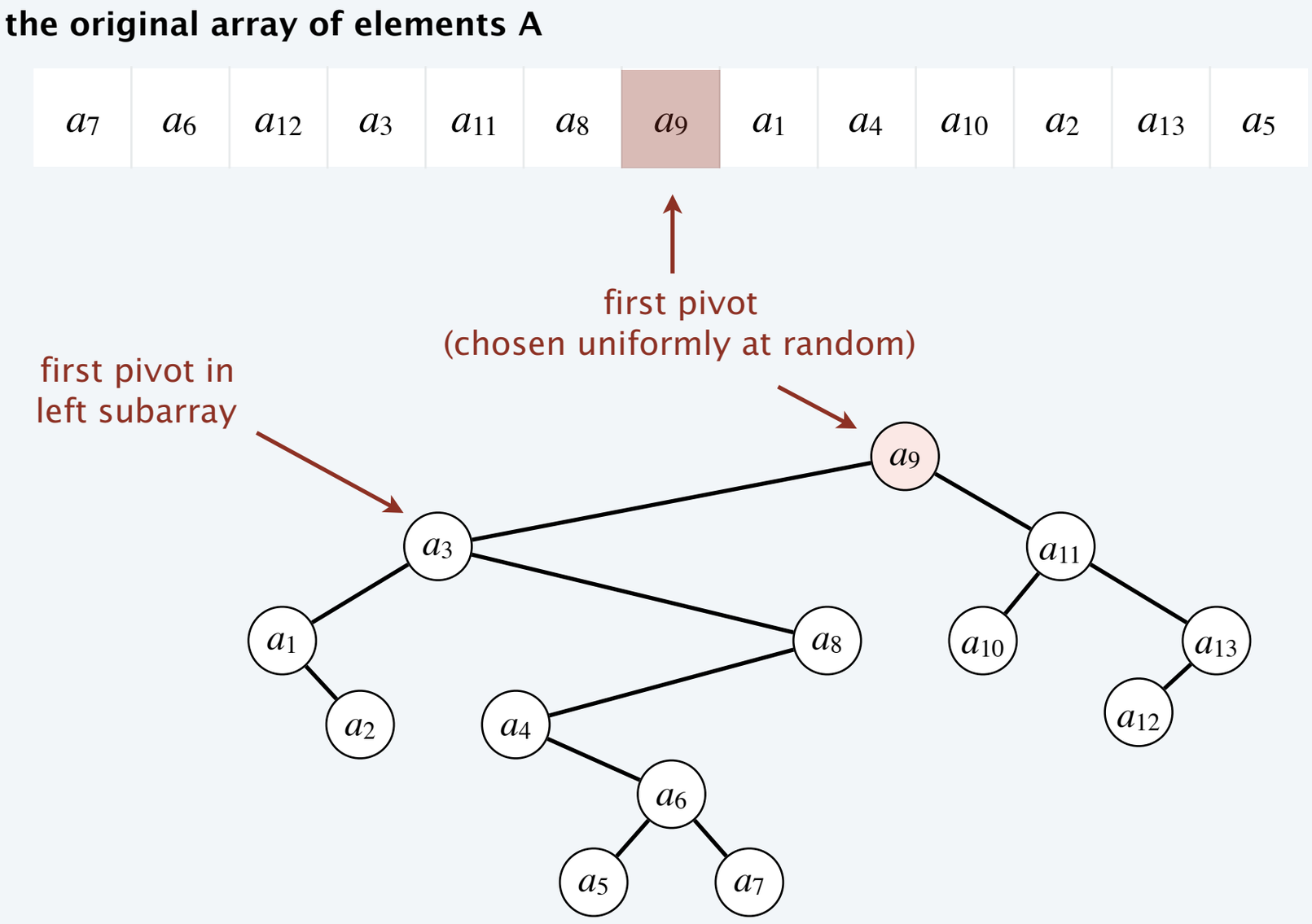
RANDOMIZED-QUICKSORT(R). ← $T(n - i - 1)$

new analysis required
(i is a random variable—depends on p)

Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \cdots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

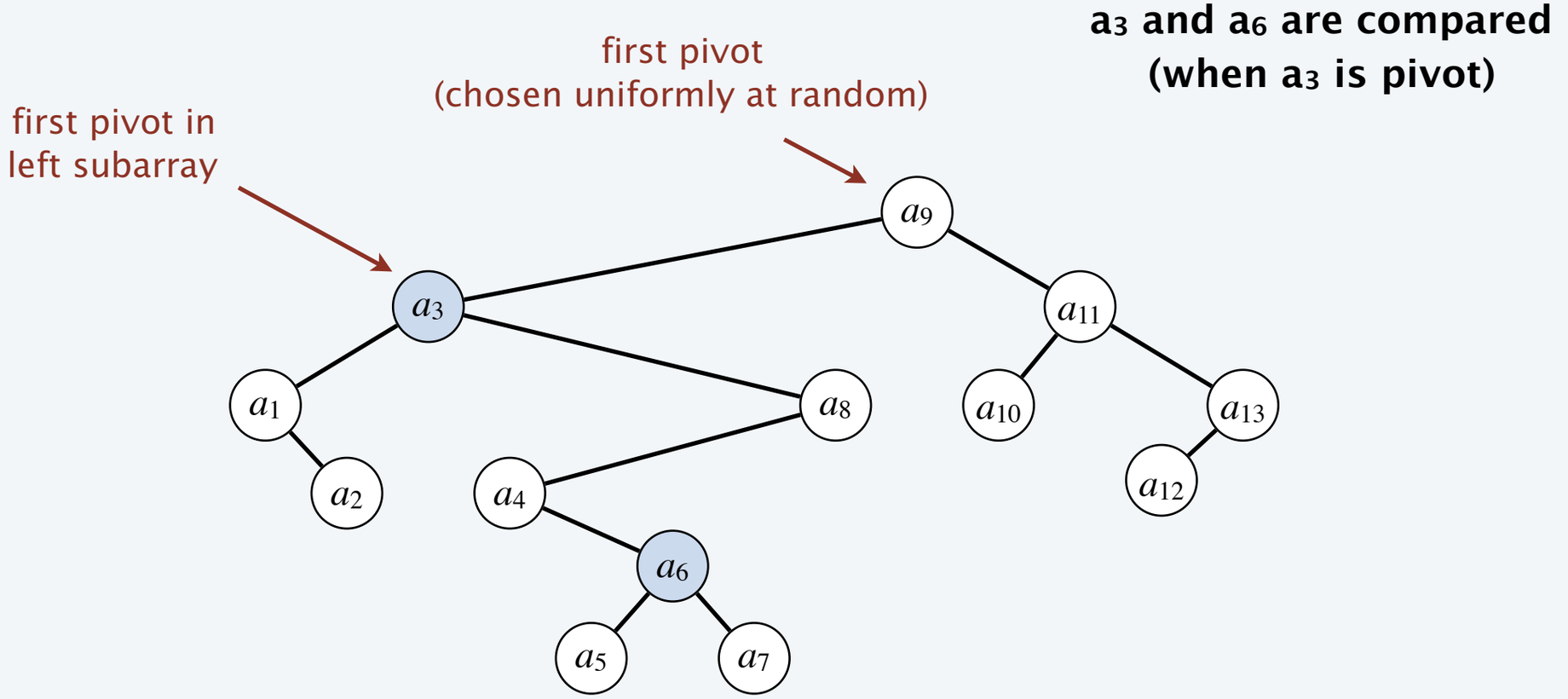


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \cdots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.

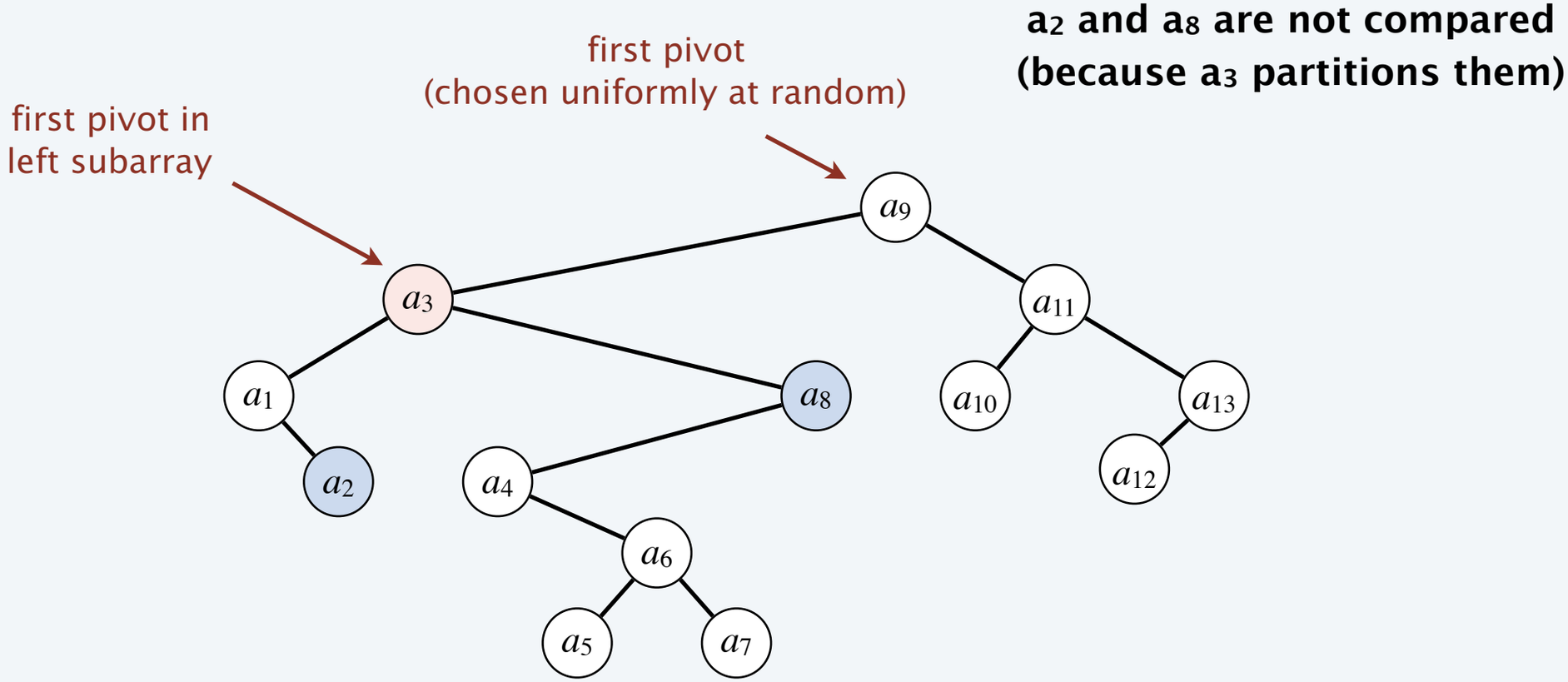


Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \cdots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.



Divide-and-conquer: quiz 2

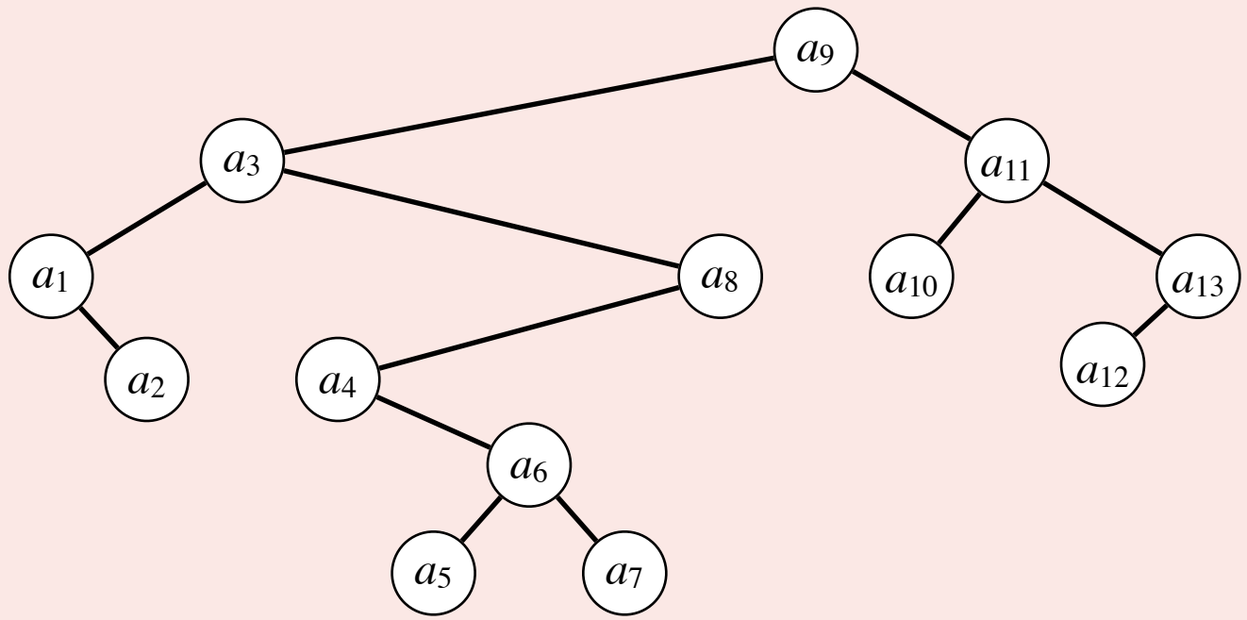
Given an array of $n \geq 8$ distinct elements $a_1 < a_2 < \cdots < a_n$, what is the probability that a_7 and a_8 are compared during randomized quicksort?

A. 0

B. $1 / n$

C. $2 / n$

D. 1



Divide-and-conquer: quiz 3

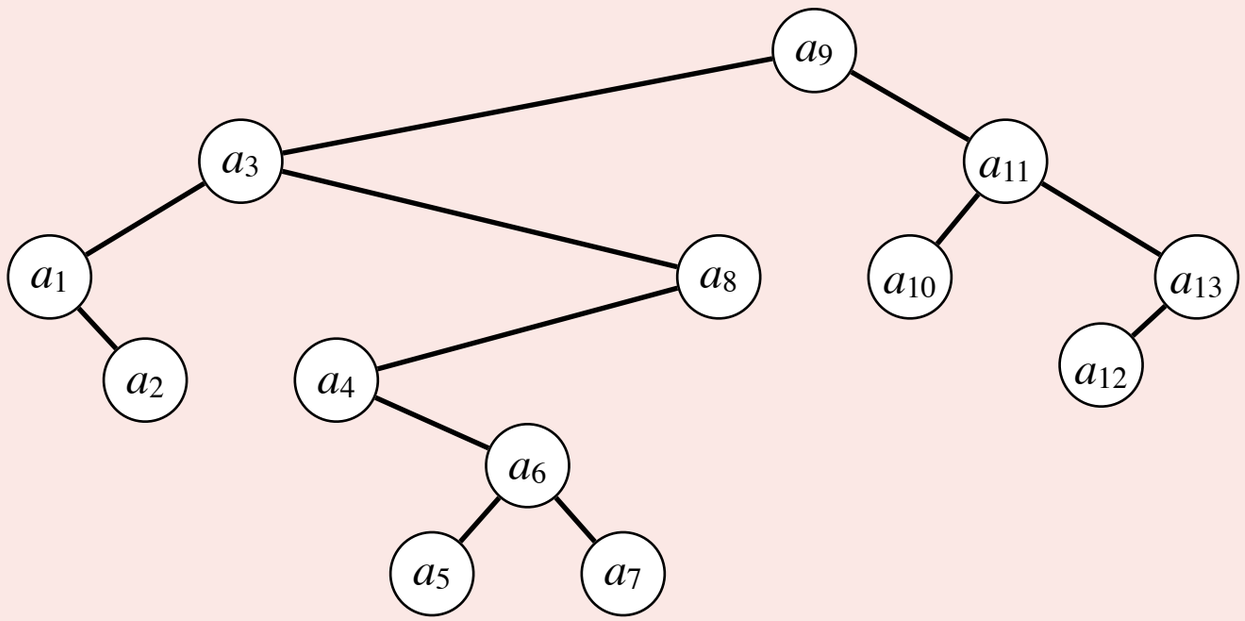
Given an array of $n \geq 2$ distinct elements $a_1 < a_2 < \cdots < a_n$, what is the probability that a_1 and a_n are compared during randomized quicksort?

A. 0

B. $1 / n$

C. $2 / n$

D. 1



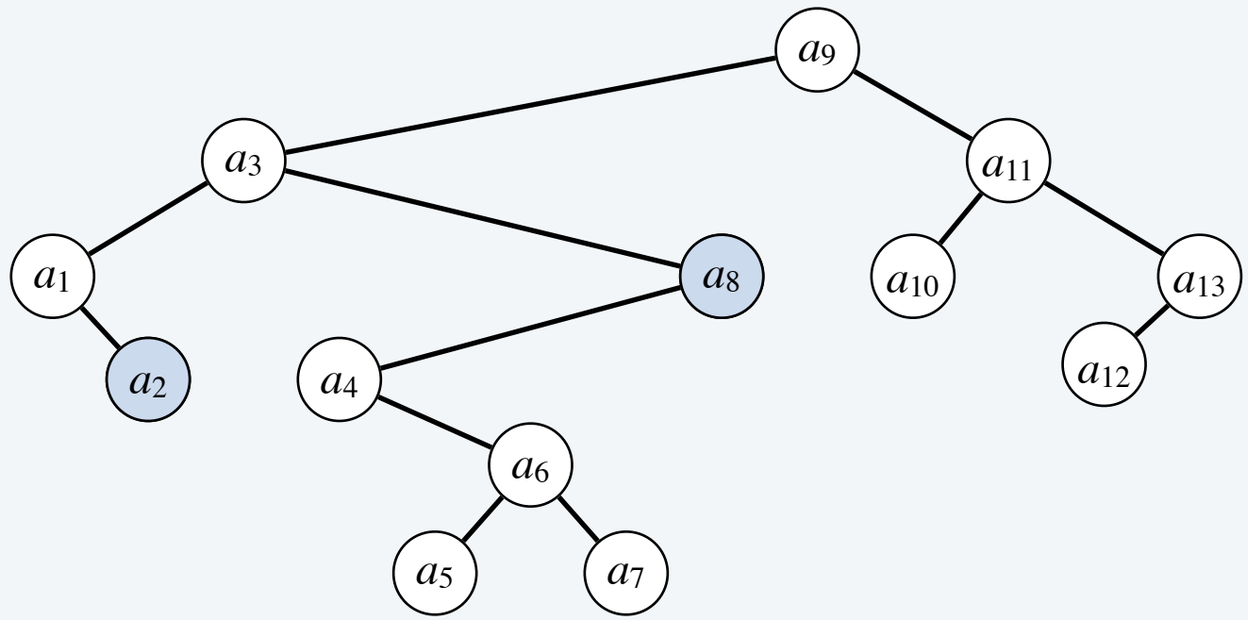
Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \cdots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.
- **Pr** [a_i and a_j are compared] $= 2 \ / \ (j - i + 1)$, where $i < j$.

Pr[a_2 and a_8 compared] = 2/7
compared iff either a_2 or a_8 is chosen as pivot before any of { a_3, a_4, a_5, a_6, a_7}



Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of n distinct elements $a_1 < a_2 < \cdots < a_n$ is $O(n \log n)$.

Pf. Consider BST representation of pivot elements.

- a_i and a_j are compared once iff one is an ancestor of the other.
- **Pr** [a_i and a_j are compared] $= 2 \; / \; (j - i + 1)$, where $i < j$.

- Expected number of compares $= \displaystyle\sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = 2 \sum_{i=1}^{n} \sum_{j=2}^{n-i+1} \frac{1}{j}$

all pairs i and j

$$\leq 2n \sum_{j=1}^{n} \frac{1}{j}$$

$$\leq 2n \, (\ln n + 1) \quad \blacksquare$$

harmonic sum

Remark. Number of compares only decreases if equal elements.

〖Example〗 Quicksort

〖Example〗 Quicksort

🕮 Deterministic Quicksort

〖Example〗 Quicksort

📖 **Deterministic Quicksort**

- ☞ $\Theta(N^2)$ worst-case running time
- ☞ $\Theta(N \log N)$ average case running time, ***assuming every input permutation is equally likely***

〖Example〗 Quicksort

📖 **Deterministic Quicksort**

- ☞ $\Theta(N^2)$ worst-case running time
- ☞ $\Theta(N \log N)$ average case running time, ***assuming*** ***every input permutation is equally likely***

☝ **How about choosing the pivot *uniformly at random*?**

〖Example〗 Quicksort

📖 **Deterministic Quicksort**

- ☞ $\Theta(N^2)$ worst-case running time
- ☞ $\Theta(N \log N)$ average case running time, *assuming*

every input permutation is equally likely

☝ How about choosing the pivot *uniformly at random*?

Central splitter := the pivot that divides the set so that each side contains at least $n/4$

〖Example〗 Quicksort

📖 **Deterministic Quicksort**

- ☞ $\Theta(N^2)$ worst-case running time
- ☞ $\Theta(N \log N)$ average case running time, *assuming every input permutation is equally likely*

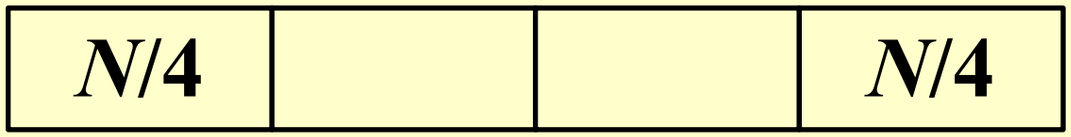
☝ How about choosing the pivot *uniformly at random*?

Central splitter := the pivot that divides the set so that each side contains **at least $n/4$**

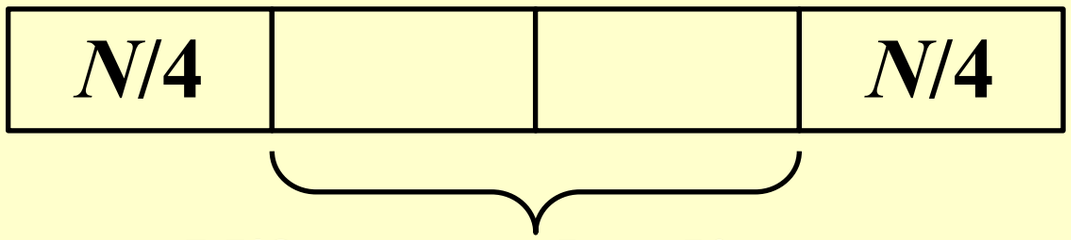
Modified Quicksort := always select a central splitter before recursions

Claim: The expected number of iterations needed until we find a central splitter is at most 2.

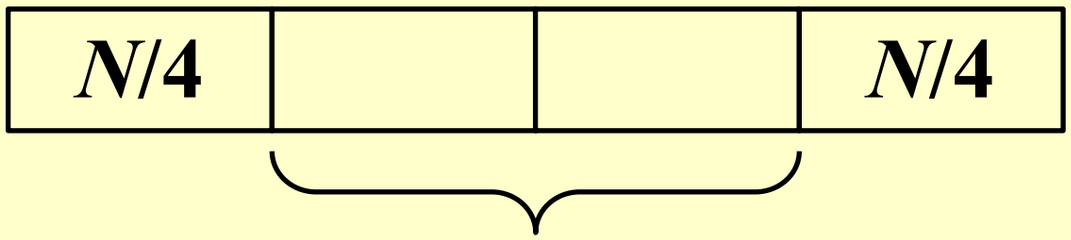
Claim: The expected number of iterations needed until we find a central splitter is at most 2.



Claim: **The expected number of iterations needed until we find a central splitter is at most 2.**

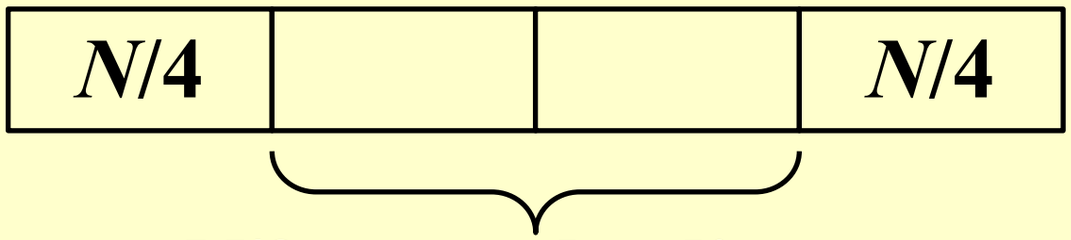


Claim: **The expected number of iterations needed until we find a central splitter is at most 2.**



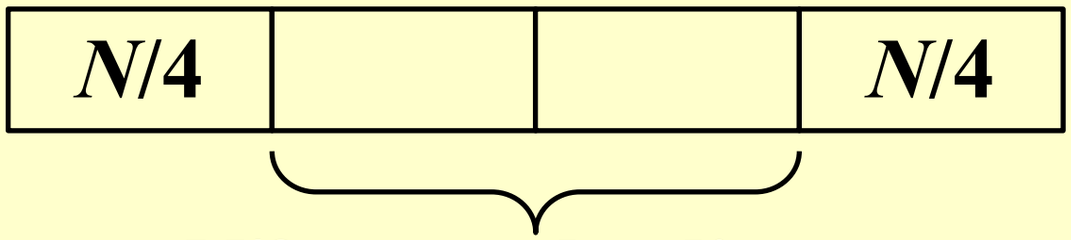
Pr[*find a central splitter*] = 1/2

Claim: **The expected number of iterations needed until we find a central splitter is at most 2.**



Pr[*find a central splitter*] = 1/2 ✓

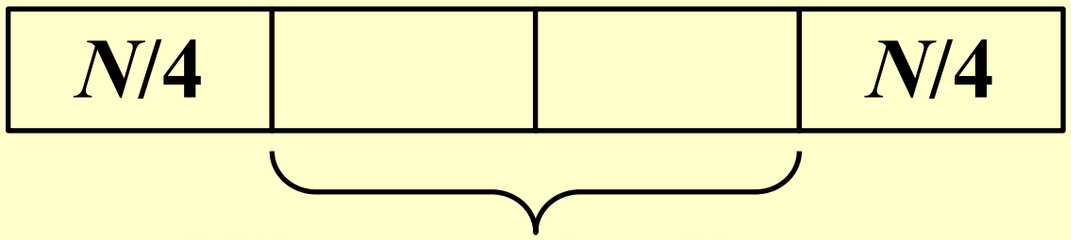
Claim: **The expected number of iterations needed until we find a central splitter is at most 2.**



Pr[*find a central splitter*] = 1/2 ✓

Type *j* : the subproblem *S* is of *type j* if $N\left(\frac{3}{4}\right)^{j+1} \le |S| \le N\left(\frac{3}{4}\right)^{j}$

Claim: **The expected number of iterations needed until we find a central splitter is at most 2.**

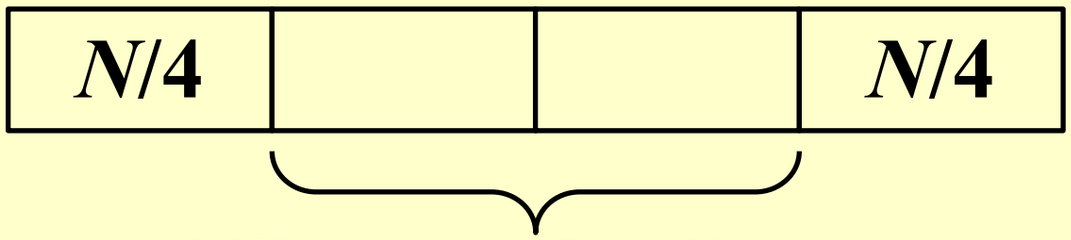


$\Pr[\textit{ find a central splitter }] = 1/2$ ✓

Type *j* : the subproblem S is of *type j* if $N\left(\frac{3}{4}\right)^{j+1} \le |S| \le N\left(\frac{3}{4}\right)^{j}$

Claim: There are at most $\left(\frac{4}{3}\right)^{j+1}$ subproblems of *type j*.

Claim: **The expected number of iterations needed until we find a central splitter is at most 2.**



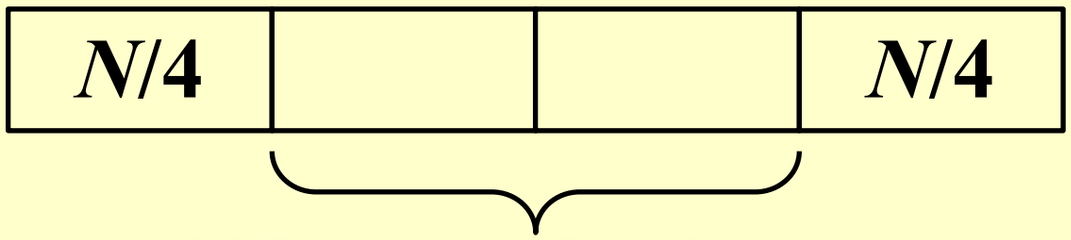
$\Pr[\textit{find a central splitter}] = 1/2$ ✓

Type *j* : the subproblem S is of ***type j*** if $N\left(\frac{3}{4}\right)^{j+1} \le |S| \le N\left(\frac{3}{4}\right)^{j}$

Claim: **There are at most** $\left(\frac{4}{3}\right)^{j+1}$ **subproblems of *type j*.**

$$E[T_{type\ j}] = O(N\left(\frac{3}{4}\right)^{j}) \times \left(\frac{4}{3}\right)^{j+1} = O(N)$$

Claim: **The expected number of iterations needed until we find a central splitter is at most 2.**



$\Pr[\textit{find a central splitter}] = 1/2$ ✓

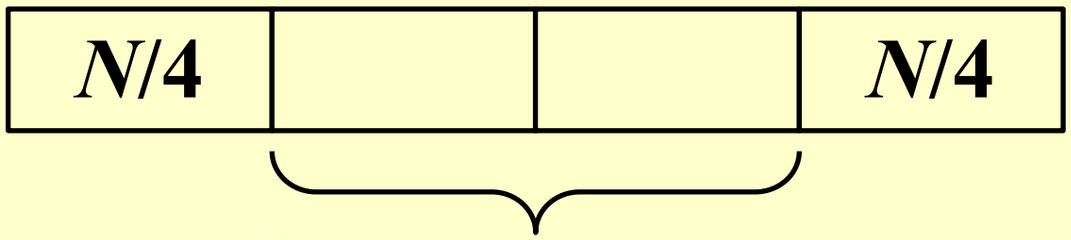
Type *j* : the subproblem S is of ***type j*** if $N\left(\frac{3}{4}\right)^{j+1} \le |S| \le N\left(\frac{3}{4}\right)^{j}$

Claim: There are at most $\left(\frac{4}{3}\right)^{j+1}$ subproblems of *type j*.

$$E[T_{type\ j}] = O\left(N\left(\frac{3}{4}\right)^{j}\right) \times \left(\frac{4}{3}\right)^{j+1} = O(N)$$

Number of different types =

Claim: **The expected number of iterations needed until we find a central splitter is at most 2.**



$\Pr[\textit{find a central splitter}] = 1/2$ ✓

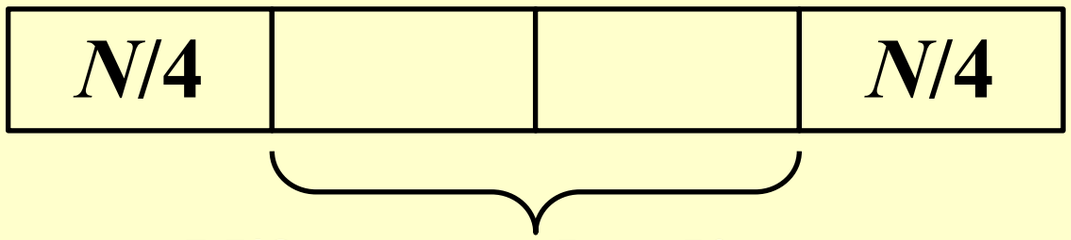
Type *j* : the subproblem S is of ***type j*** if $N\left(\frac{3}{4}\right)^{j+1} \le |S| \le N\left(\frac{3}{4}\right)^{j}$

Claim: There are at most $\left(\frac{4}{3}\right)^{j+1}$ subproblems of *type j*.

$$E[T_{type\ j}] = O(N\left(\frac{3}{4}\right)^{j}) \times \left(\frac{4}{3}\right)^{j+1} = O(N)$$

Number of different types = $\log_{4/3} N = O(\log N)$

Claim: **The expected number of iterations needed until we find a central splitter is at most 2.**



$\Pr[\textit{find a central splitter}] = 1/2$ ✓

Type *j* : the subproblem S is of ***type j*** if $N\left(\frac{3}{4}\right)^{j+1} \le |S| \le N\left(\frac{3}{4}\right)^{j}$

Claim: There are at most $\left(\frac{4}{3}\right)^{j+1}$ subproblems of *type j*.

$$E[T_{type\ j}] = O\left(N\left(\frac{3}{4}\right)^{j}\right) \times \left(\frac{4}{3}\right)^{j+1} = O(N)$$

Number of different types $= \log_{4/3} N = O(\log N)$

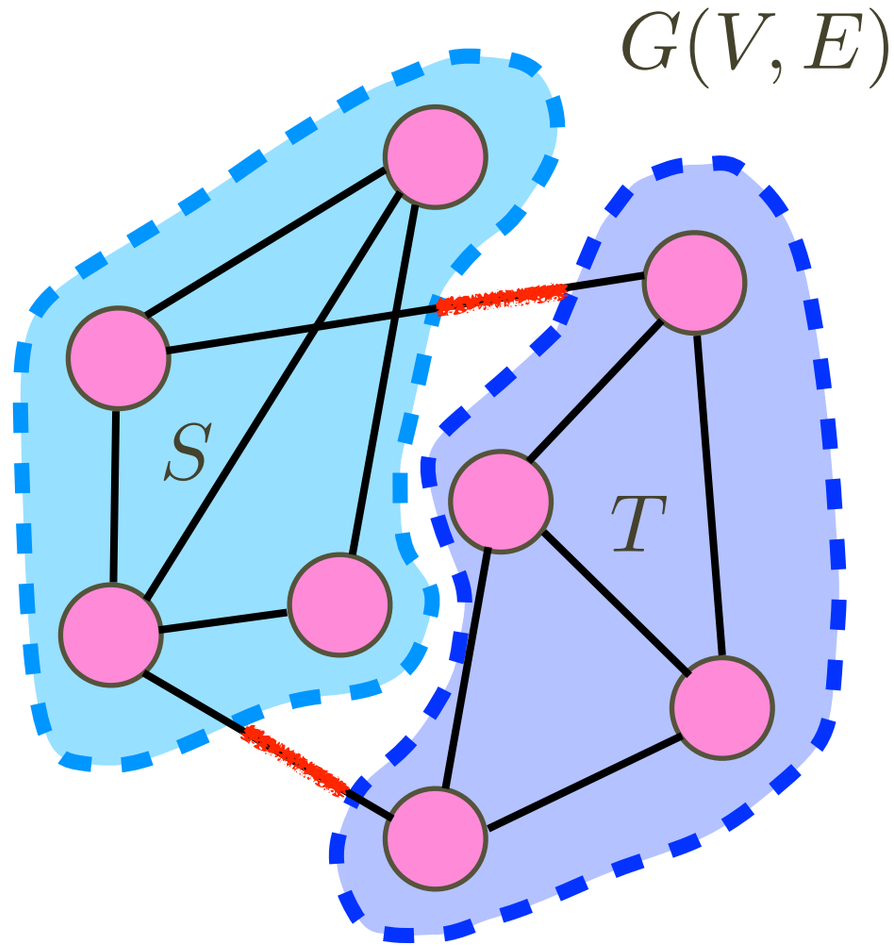
$\Big\}\ O(N \log N)$

Randomized Algorithms

- Hiring problem: probabilistic analysis
- Randomized hiring algorithm
- Randomized quicksort
- Min-cut and max-cut
- Take-home messages

Min-Cut

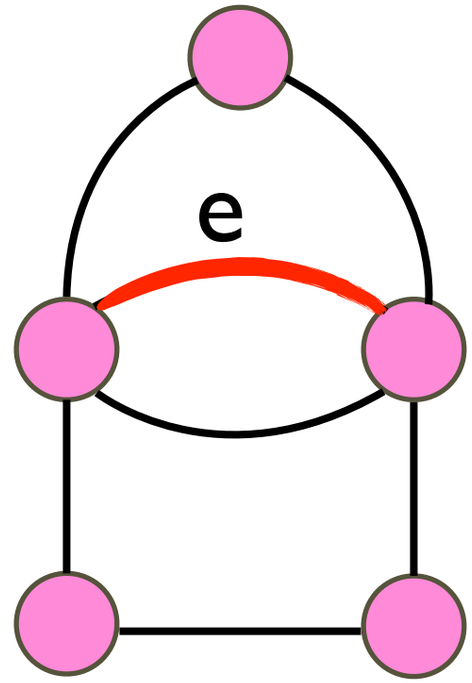
- Partition V into two parts: S and T
- Minimize the cut $E(S,T)$
- deterministic algorithm:
  - max-flow min-cut
  - best known upper bound: $O(mn + n^2 \log n)$



$$E(S,T) = \{uv \in E \mid u \in S, v \in T\}$$

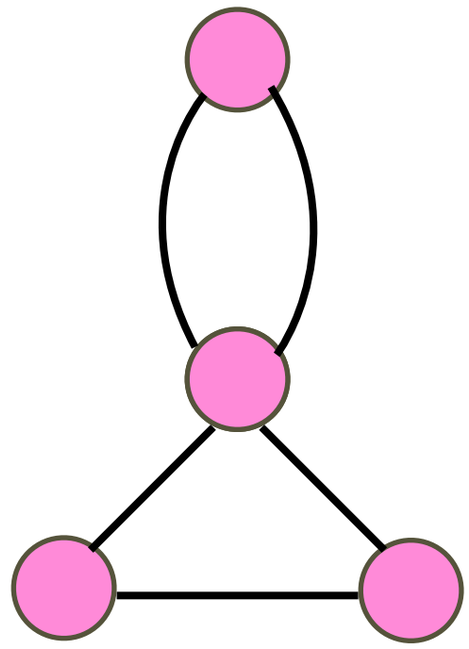
Contraction

- multigraph $G(V, E)$
- multigraph: allow parallel edges
- for an edge e, $\text{contract}(e)$ merges the two endpoints.



Contraction

- multigraph $G(V, E)$
- multigraph: allow parallel edges
- for an edge e, $\text{contract}(e)$ merges the two endpoints.



Karger's min-cut Algorithm

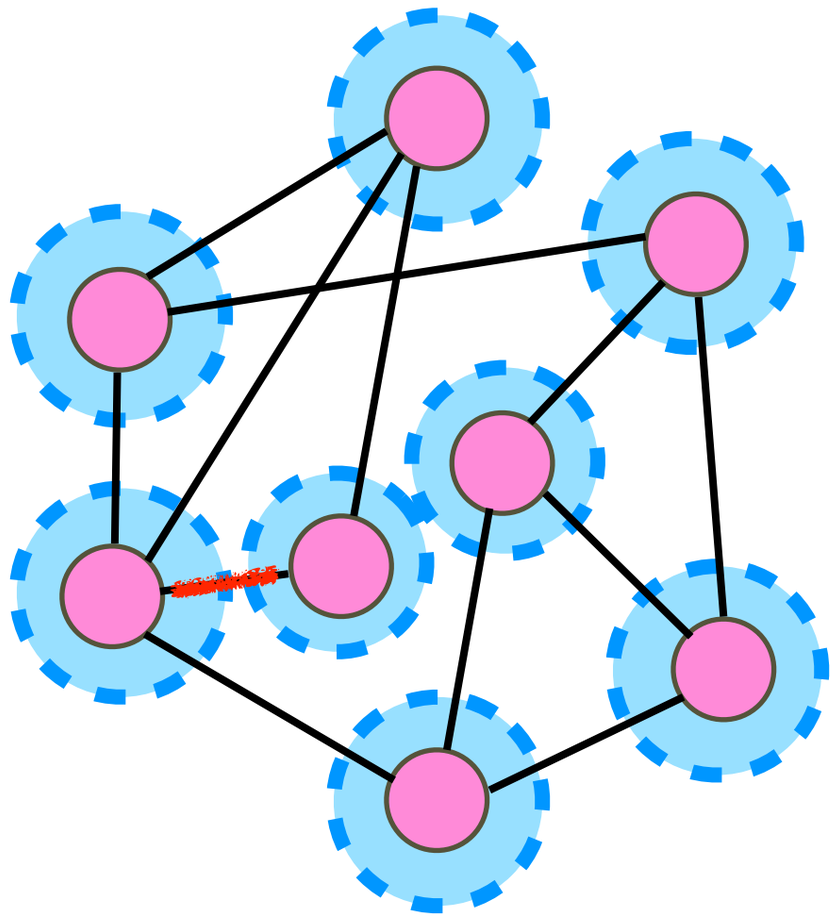
MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;



Karger's min-cut Algorithm

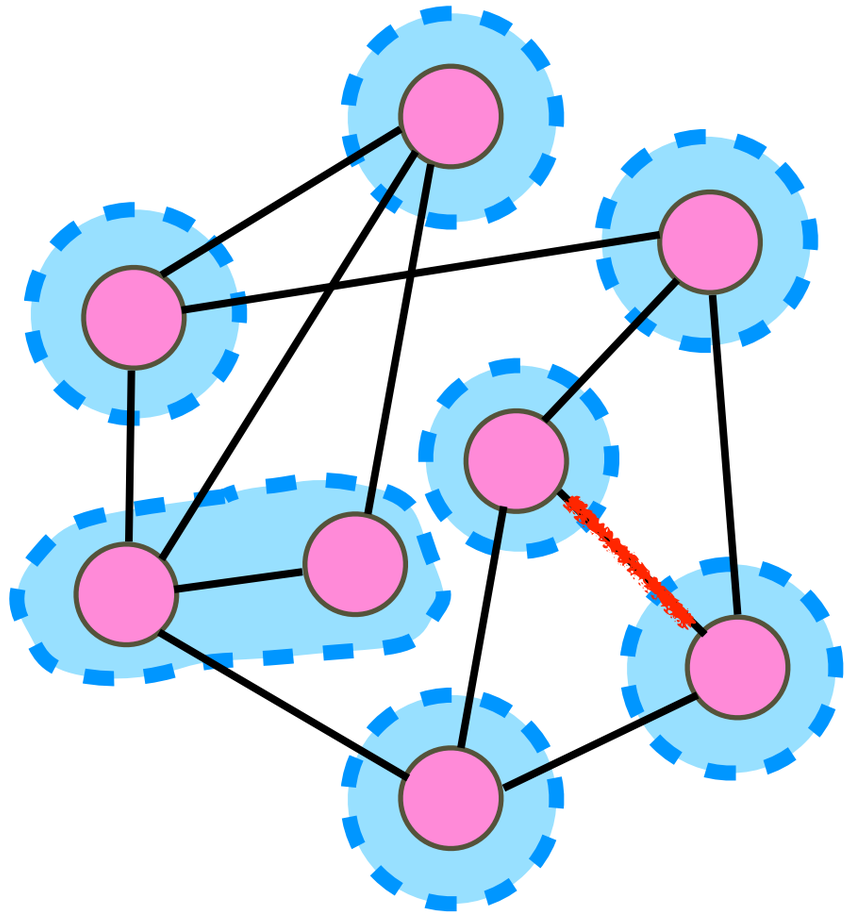
MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

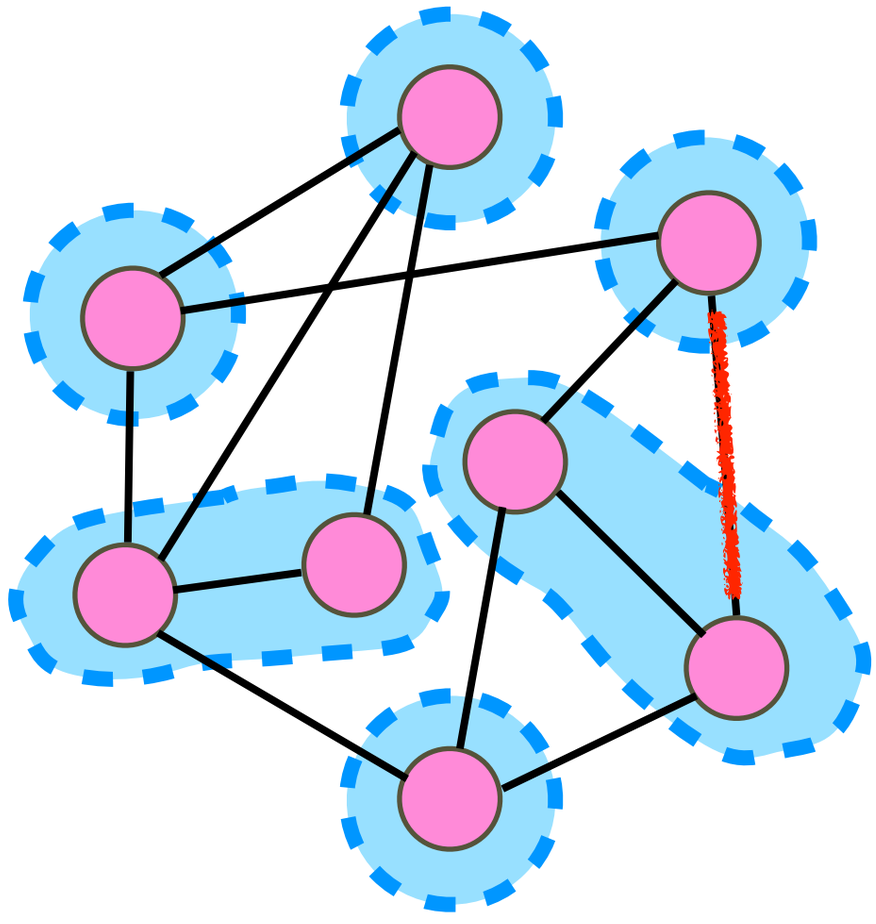
return remaining edges;



Karger's min-cut Algorithm

MinCut (multigraph $G(V,E)$)

while $\|V\|>2$ do
choose a uniform $e \in E$;
contract(e);
return remaining edges;



Karger's min-cut Algorithm

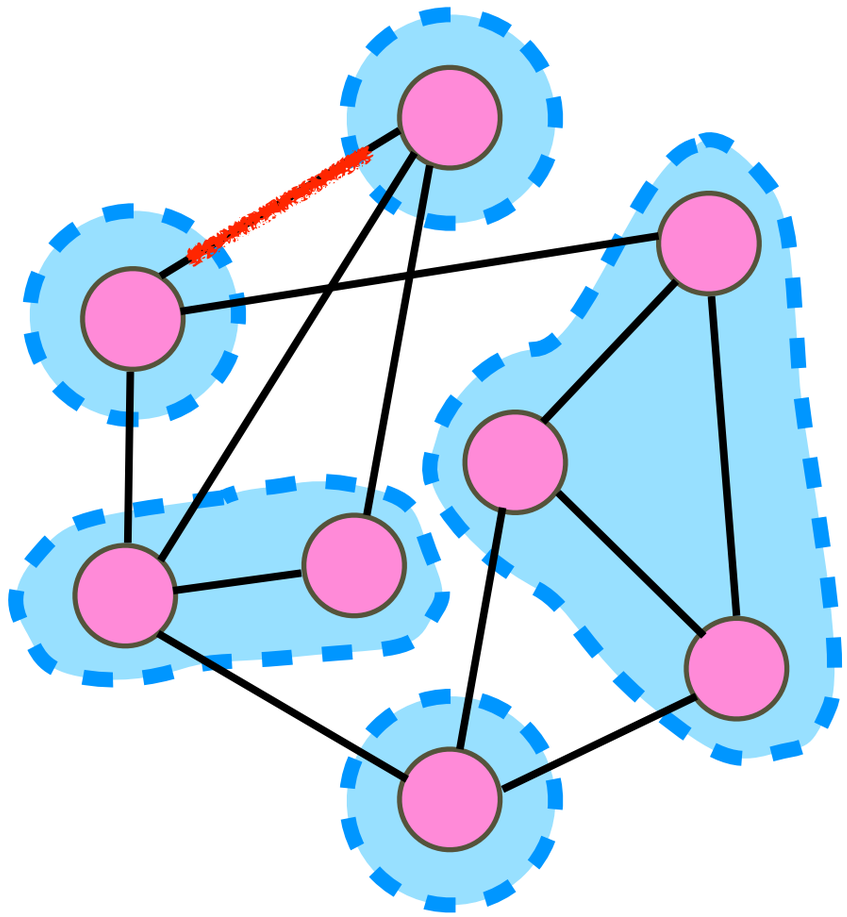
MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;



Karger's min-cut Algorithm

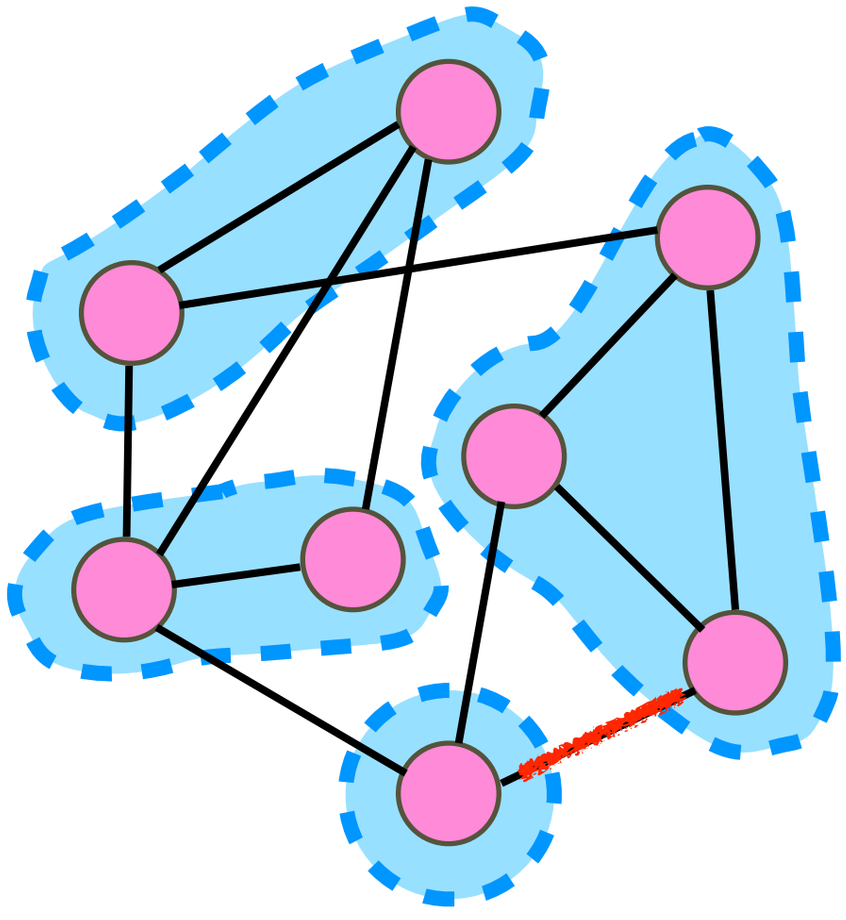
MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;



Karger's min-cut Algorithm

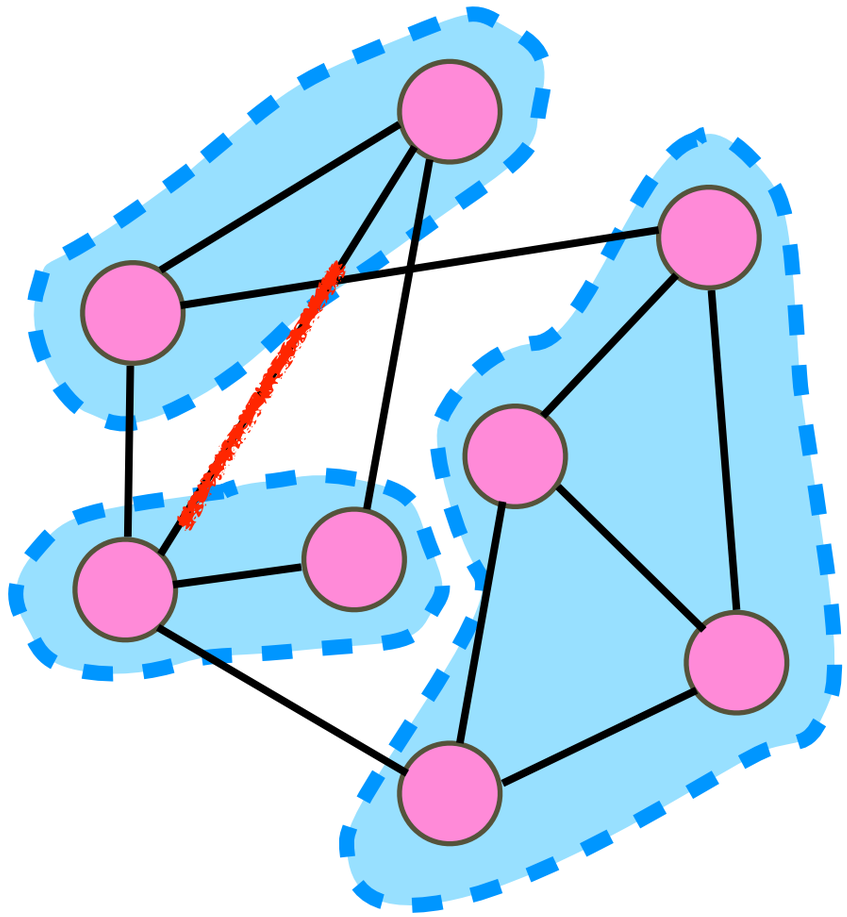
MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

choose a uniform $e \in E$;

contract(e);

return remaining edges;



Karger's min-cut Algorithm

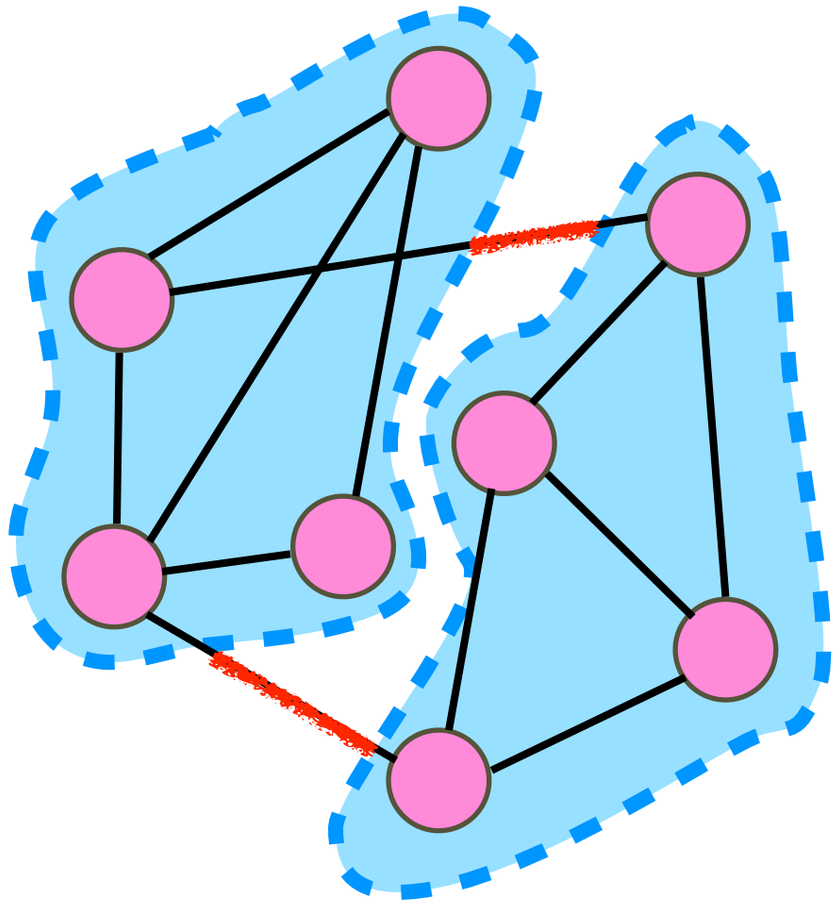
MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

choose a uniform $e \in E$;

contract(e);

return remaining edges;



edges returned

Karger's min-cut Algorithm

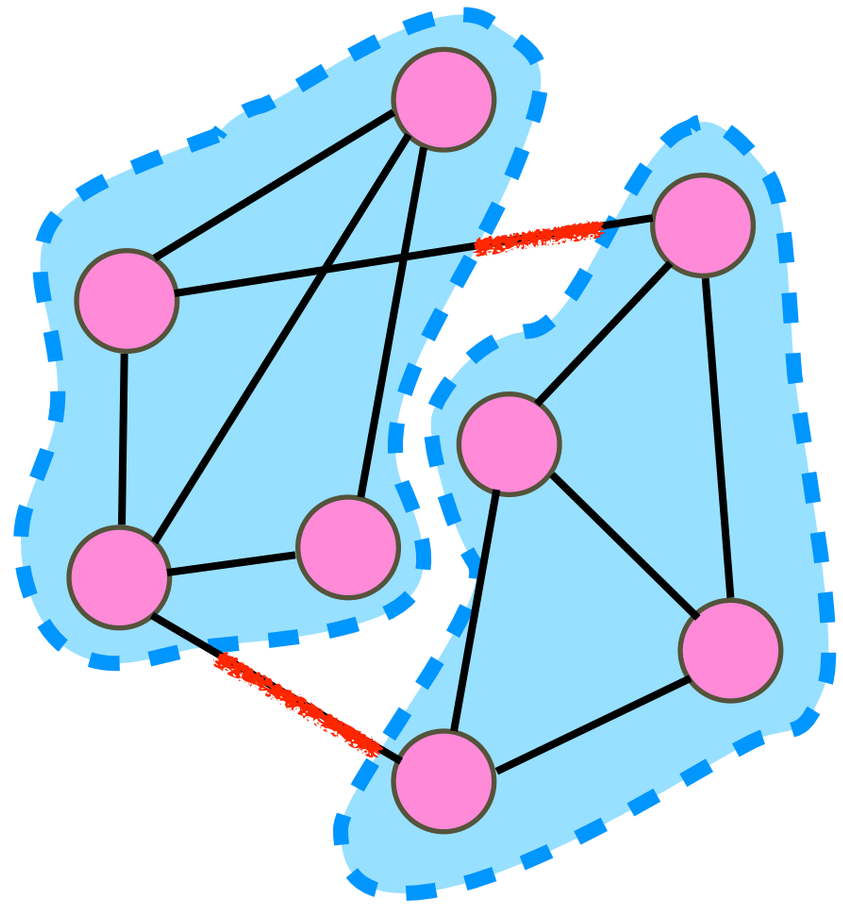
MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

choose a uniform $e \in E$;

contract(e);

return remaining edges;



edges returned

Each contract operation takes O(n) time.

MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;

MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;

Theorem (Karger 1993)**:**

Pr[a min-cut is returned] $\geq \frac{2}{n(n-1)}$

MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;

Theorem (Karger 1993)**:**

$$\Pr[\text{ a min-cut is returned }] \geq \frac{2}{n(n-1)}$$

repeat independently
for $n(n\text{-}1)/2$ times
and return the smallest cut

MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;

Theorem (Karger 1993)**:**

$$\Pr[\text{ a min-cut is returned }] \geq \frac{2}{n(n-1)}$$

repeat independently
for $n(n\text{-}1)/2$ times
and return the smallest cut

$$\Pr[\text{ fail to finally return a min-cut }]$$

$$= \Pr[\text{ fail to construct a min-cut in one trial }]^{n(n-1)/2}$$

$$\leq \left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)/2} < \frac{1}{\mathrm{e}}$$

MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;

suppose $e_1, e_2, \ldots, e_{n-2}$

are contracted edges

initially: $G_1 = G$

i-th round:

$$G_i = \mathrm{contract}(G_{i-1}, e_{i-1})$$

```
MinCut ( multigraph G(V,E) )
while |V|>2 do
  choose a uniform e ∈E ;
  contract(e);
return remaining edges;
```

suppose $e_1, e_2, \ldots, e_{n-2}$

are contracted edges

initially: $G_1 = G$

i-th round:

$$G_i = \text{contract}(G_{i-1}, e_{i-1})$$

$$\left.\begin{array}{l} C \text{ is a min-cut in } G_{i-1} \\ e_{i-1} \notin C \end{array}\right\} \Rightarrow C \text{ is a min-cut in } G_i$$

MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;

suppose $e_1, e_2, \ldots, e_{n-2}$

are contracted edges

initially: $G_1 = G$

i-th round:

$$G_i = \text{contract}(G_{i-1}, e_{i-1})$$

C is a min-cut in G_{i-1}

$e_{i-1} \notin C$

$\Rightarrow$ C is a min-cut in G_i

C: a min-cut of G

$$\Pr[C \text{ is returned}] \geq \Pr[e_1, e_2, \ldots, e_{n-2} \notin C]$$

chain rule:

$$= \prod_{i=1}^{n-2} \Pr[e_i \notin C \mid e_1, e_2, \ldots, e_{i-1} \notin C]$$

suppose $e_1, e_2, \ldots, e_{n-2}$ are contracted edges

initially: $G_1 = G$ *i*-th round: $G_i = \text{contract}(G_{i-1}, e_{i-1})$

C is a min-cut in G_{i-1}
$e_{i-1} \notin C$
⇒ C is a min-cut in G_i

C: a min-cut of G

$$\Pr[C \text{ is returned}] \geq \prod_{i=1}^{n-2} \Pr[e_i \notin C \mid e_1, e_2, \ldots, e_{i-1} \notin C]$$

C is a min-cut in G_i

suppose $e_1, e_2, \ldots, e_{n-2}$ are contracted edges

initially: $G_1 = G$ *i*-th round: $G_i = \text{contract}(G_{i-1}, e_{i-1})$

C is a min-cut in G_{i-1}
$e_{i-1} \notin C$
⇒ C is a min-cut in G_i

C: a min-cut of G

$$\Pr[C \text{ is returned}] \geq \prod_{i=1}^{n-2} \Pr[e_i \notin C \mid e_1, e_2, \ldots, e_{i-1} \notin C]$$

$e_1, e_2, \ldots, e_{i-1} \notin C$ ⇒ C is a min-cut in G_i

C is a min-cut in $G(V, E)$
⇒ $|E| \geq \frac{1}{2}|C||V|$

suppose $e_1, e_2, \ldots, e_{n-2}$ are contracted edges

initially: $G_1 = G$ *i*-th round: $G_i = \text{contract}(G_{i-1}, e_{i-1})$

C is a min-cut in G_{i-1}
$e_{i-1} \notin C$
⇒ C is a min-cut in G_i

C: a min-cut of G

$$\Pr[C \text{ is returned}] \geq \prod_{i=1}^{n-2} \Pr[e_i \notin C \mid e_1, e_2, \ldots, e_{i-1} \notin C]$$

$e_1, e_2, \ldots, e_{i-1} \notin C$ ⇒ C is a min-cut in G_i

C is a min-cut in $G(V, E)$
⇒ $|E| \geq \frac{1}{2}|C||V|$

Proof:

min-degree of $G \geq |C|$

suppose $e_1, e_2, \ldots, e_{n-2}$ are contracted edges

initially: $G_1 = G$ *i*-th round: $G_i = \text{contract}(G_{i-1}, e_{i-1})$

C is a min-cut in G_{i-1}
$e_{i-1} \notin C$
⇒ C is a min-cut in G_i

C: a min-cut of G

$$\Pr[C \text{ is returned}] \geq \prod_{i=1}^{n-2} \Pr[e_i \notin C \mid e_1, e_2, \ldots, e_{i-1} \notin C]$$

$e_1, e_2, \ldots, e_{i-1} \notin C$ ⇒ C is a min-cut in G_i

C is a min-cut in $G(V, E)$
⇒ $|E| \geq \frac{1}{2}|C||V|$

$$\geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{(n-i+1)}\right)$$

Proof:

min-degree of $G \geq |C|$

suppose $e_1, e_2, \ldots, e_{n-2}$ are contracted edges

initially: $G_1 = G$ *i*-th round: $G_i = \text{contract}(G_{i-1}, e_{i-1})$

C is a min-cut in G_{i-1}
$e_{i-1} \notin C$
$\Rightarrow$ C is a min-cut in G_i

C: a min-cut of G

$$\Pr[C \text{ is returned}] \geq \prod_{i=1}^{n-2} \Pr[e_i \notin C \mid e_1, e_2, \ldots, e_{i-1} \notin C]$$

C is a min-cut in G_i

C is a min-cut in $G(V, E)$
$\Rightarrow$ $|E| \geq \frac{1}{2}|C||V|$

$$\geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{(n-i+1)}\right)$$

Proof:

min-degree of $G \geq |C|$

$$= \prod_{i=1}^{n-2} \frac{n-i-1}{n-i+1} = \frac{2}{n(n-1)}$$

MinCut (multigraph $G(V,E)$)

while $|V|>2$ do

  choose a uniform $e \in E$;

  contract(e);

return remaining edges;

Theorem (Karger 1993)**:**

For any min-cut C,

$$\Pr[C \text{ is returned}] \geq \frac{2}{n(n-1)}$$

```
MinCut ( multigraph G(V,E) )
while |V|>2 do
  choose a uniform e ∈E ;
  contract(e);
return remaining edges;
```

Theorem (Karger 1993)**:**

For any min-cut C,

$$\Pr[C \text{ is returned}] \geq \frac{2}{n(n-1)}$$

running time: $O(n^2)$

repeat *independently* for $O(n^2 \log n)$ times

returns a min-cut with probability $1\text{-}O(1/n)$

total running time: $O(n^4 \log n)$

Number of Min-Cuts

Theorem (Karger 1993)**:**

For any min-cut C,

$$\Pr[C \text{ is returned}] \geq \frac{2}{n(n-1)}$$

Corollary

The number of distinct min-cuts
in a graph of n vertices is at most $n(n\text{-}1)/2$.

Number of Min-Cuts

Theorem (Karger 1993)**:**

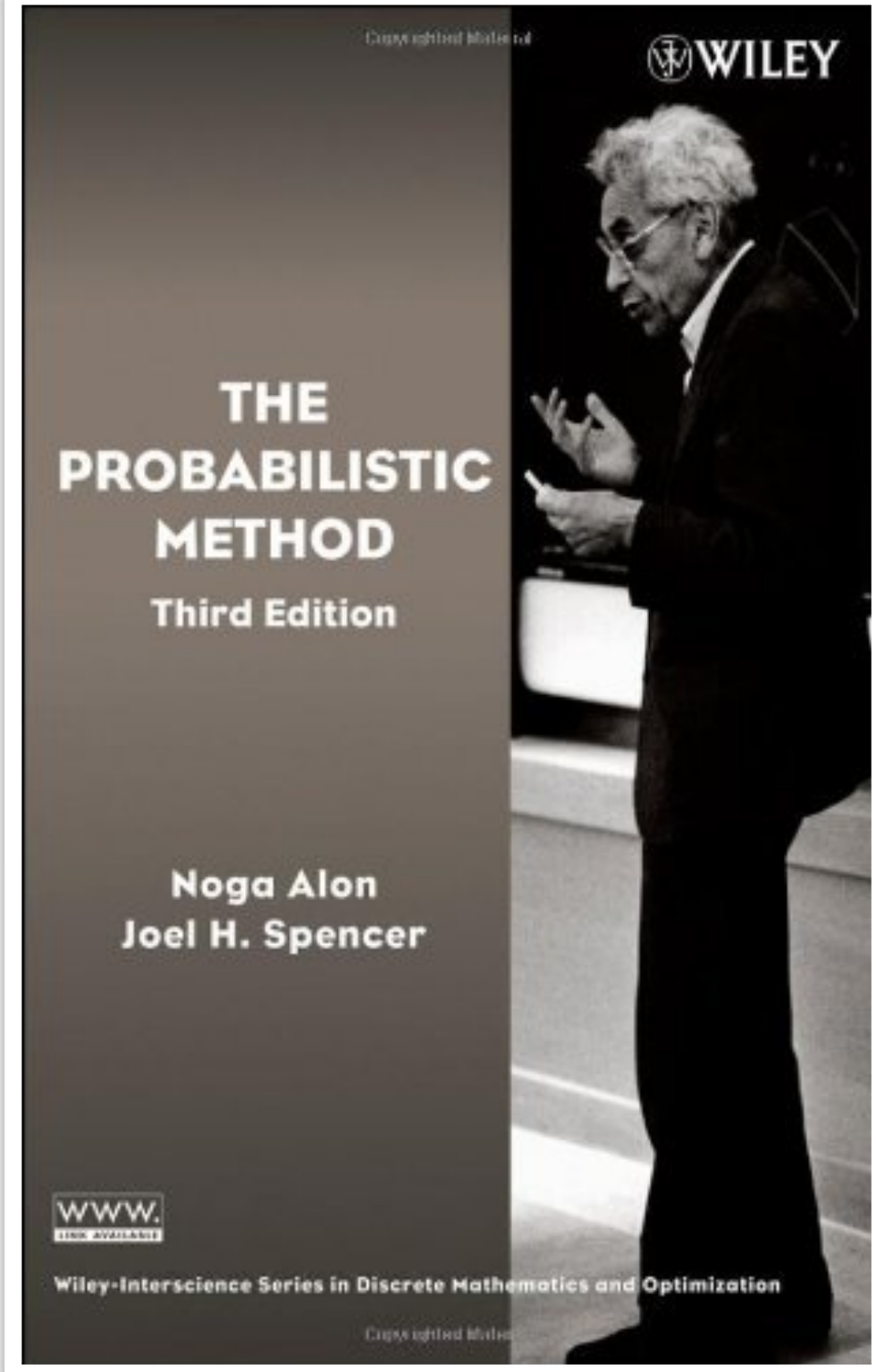
For any min-cut C,

$$\Pr[C \text{ is returned}] \geq \frac{2}{n(n-1)}$$

Corollary

The number of distinct min-cuts
in a graph of n vertices is at most $n(n\text{-}1)/2$.

No randomness in the corollary at all.
An example of the probabilistic method.


WILEY
THE PROBABILISTIC METHOD
Third Edition
Noga Alon
Joel H. Spencer
WWW.
Wiley-Interscience Series in Discrete Mathematics and Optimization

An Observation

MinCut (multigraph $G(V,E)$)

while $|V|> t$ do

choose a uniform $e \in E$;

contract(e);

return remaining edges;

C: a min-cut of G

$$\Pr[e_1, \ldots, e_{n-t} \notin C] = \prod_{i=1}^{n-t} \Pr[e_i \notin C \mid e_1, \ldots, e_{i-1} \notin C]$$

$$\geq \prod_{i=1}^{n-t} \frac{n-i-1}{n-i+1} = \frac{t(t-1)}{n(n-1)}$$

only getting bad when t is small

Fast Min-Cut

```
Contract ( G, t )
while |V|> t do
  choose a uniform e ∈E ;
  contract(e);
return remaining edges;
```

```
FastCut ( G )
if |V|≤6 then return a min-cut by brute force;
else:  (t to be fixed later)
  G1= Contract(G,t);
  G2= Contract(G,t);          (independently)
    return min{FastCut(G1), FastCut(G2)};
```

FastCut (G)

if $|V| \le 6$ then return a min-cut by brute force;

else: (t to be fixed later)

G_1= Contract(G,t);

G_2= Contract(G,t); (independently)

return min{FastCut(G_1), FastCut(G_2)};

C: a min-cut in G

A: no edge in C is contracted during Contract(G,t)

$$\Pr[A] = \prod_{i=1}^{n-t} \Pr[e_i \notin C \mid e_1, \ldots, e_{i-1} \notin C]$$

$$\geq \prod_{i=1}^{n-t} \frac{n-i-1}{n-i+1} \quad \geq \frac{t(t-1)}{n(n-1)} \quad \geq \left(\frac{t-1}{n-1}\right)^2$$

FastCut (G)

if $|V| \le 6$ then return a min-cut by brute force;

else: (t to be fixed later)

G_1= Contract(G,t);

G_2= Contract(G,t); (independently)

return min{FastCut(G_1), FastCut(G_2)};

C: a min-cut in G set $t = \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil$

A: no edge in C is contracted during Contract(G,t)

$$\Pr[A] \ge \left(\frac{t-1}{n-1}\right)^2 \ge \frac{1}{2}$$

FastCut (G)

if $|V|\le 6$ then return a min-cut by brute force;

else: (t to be fixed later)

G_1= Contract(G,t);

G_2= Contract(G,t); (independently)

return min{FastCut(G_1), FastCut(G_2)};

C: a min-cut in G set $t = \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil$

A: no edge in C is contracted during Contract(G,t)

$$\Pr[A] \ge \left(\frac{t-1}{n-1}\right)^2 \ge \frac{1}{2}$$

$$p(n) = \min_{G:|V|=n} \Pr[\,\text{FastCut}(G) \text{ returns a mincut in } G\,]$$

succeeds

$$\ge 1 - (1 - \Pr[A]\Pr[\text{FastCut}(G_1) \text{ succeeds} \mid A])^2$$

$$\ge 1 - \left(1 - \left(\frac{t-1}{n-1}\right)^2 p\,(t)\right)^2 \ge p\left(\left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil\right) - \frac{1}{4} p\left(\left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil\right)^2$$

FastCut (G)

if $|V| \le 6$ then return a min-cut by brute force;

else: set $t = \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil$

G_1= Contract(G,t);

G_2= Contract(G,t);

(independently)

return min{FastCut(G_1), FastCut(G_2)};

$$p(n) = \min_{G:|V|=n} \Pr[\,\text{FastCut}(G) \text{ returns a mincut in } G\,]$$

$$\geq p\left(\left\lceil 1 + \tfrac{n}{\sqrt{2}} \right\rceil\right) - \tfrac{1}{4} p\left(\left\lceil 1 + \tfrac{n}{\sqrt{2}} \right\rceil\right)^2$$

FastCut (G)

if $|V| \le 6$ then return a min-cut by brute force;

else: set $t = \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil$

G_1= Contract(G,t);

G_2= Contract(G,t);

(independently)

return min{FastCut(G_1), FastCut(G_2)};

$$p(n) = \min_{G:|V|=n} \Pr[\,\text{FastCut}(G) \text{ returns a mincut in } G\,]$$

$$\geq p\left(\left\lceil 1 + \tfrac{n}{\sqrt{2}} \right\rceil\right) - \tfrac{1}{4} p\left(\left\lceil 1 + \tfrac{n}{\sqrt{2}} \right\rceil\right)^2$$

by induction: $p(n) = \Omega\left(\frac{1}{\log n}\right)$

running time: $T(n) = 2T\left(\left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil\right) + O(n^2)$

by induction: $T(n) = O(n^2 \log n)$

FastCut (G)

if $|V| \leq 6$ then return a min-cut by brute force;

else: set $t = \left\lceil 1 + \frac{n}{\sqrt{2}} \right\rceil$

G_1= Contract(G,t);

G_2= Contract(G,t);

(independently)

return min{FastCut(G_1), FastCut(G_2)};

Theorem (Karger-Stein 1996)**:**

FastCut runs in time $O(n^2 \log n)$ and returns a min-cut with probability $\Omega(1/\log n)$.

repeat *independently* for $O((\log n)^2)$ times

total running time: $O(n^2 \log^3 n)$

returns a min-cut with probability $1\text{-}O(1/n)$

Take-Home Messages

- Probabilistic analysis:
  - Considering probability distributions over inputs for average analysis.
- Randomized algorithm:
  - Injection of randomness in running steps.
  - Relaxation by allowing uncertainty in correctness or running time.

Thanks for your attention!
Discussions?

Reference

Introduction to algorithms, 4th edition, Section 5.

Algorithm design: Chap. 13.

Slides from Kevin Wayne: https://www.cs.princeton.edu/~wayne/kleinberg-tardos/

Yitong Yin's lecture:

https://tcs.nju.edu.cn/wiki/index.php?title=%E9%AB%98%E7%BA%A7%E7%AE%97%E6%B3%95_(Fall_2020)/Min-Cut_and_Max-Cut

https://tcs.nju.edu.cn/slides/aa2020/Cut.pdf